PENGEMBANGAN SISTEM DETEKSI HUNIAN PARKIR MENGGUNAKAN METODE CONVOLUTIONAL NEURAL NETWORK

Disusun Oleh:

N a m a : Fatih Assidhiqi

NIM : 17523220

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2021

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGEMBANGAN SISTEM DETEKSI HUNIAN PARKIR MENGGUNAKAN METODE CONVOLUTIONAL NEURAL NETWORK

TUGAS AKHIR

Disusun Oleh:

N a m a : Fatih Assidhiqi

NIM : 17523220

Yogyakarta, 06 Mei 2021

Pembimbing,

(Dr. Ing. Ridho Rahmadi, S. Kom., M. Sc.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGEMBANGAN SISTEM DETEKSI HUNIAN PARKIR MENGGUNAKAN METODE CONVOLUTIONAL NEURAL NETWORK

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 06 Mei 2021

Tim Penguji

Dr. Ing. Ridho Rahmadi, S. Kom., M. Sc.

Anggota 1

Taufiq Hidayat, S.T., M.C.S.

Anggota 2

Septia Rani, S.T., M.Cs.

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Fatih Assidhiqi

NIM : 17523220

Tugas akhir dengan judul:

PENGEMBANGAN SISTEM DETEKSI HUNIAN PARKIR MENGGUNAKAN METODE CONVOLUTIONAL NEURAL NETWORK

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 06 Mei 2021

(Fatih Assidhiqi)

HALAMAN PERSEMBAHAN

Alhamdulillah, segala puji dan syukur bagi Allah SWT. Karena tugas akhir yang merupakan bagian dari perjalanan hidup dan ibadah saya kepada Allah SWT akhirnya telah selesai. Dengan segala kendala yang dihadapi, Alhamdulillah tugas ini selesai. Tugas akhir saya persembahkan untuk:

1. Bapak Muanis dan ibu Dyah Kristianingsih yang saya cintai, saya kasihi dan juga saya hormati. Terimakasih atas segala bentuk perhatian, dukungan serta doa yang telah diberikan sehingga saya dapat menyelesaikan tugas akhir saya. Mungkin ini belum cukup untuk membalas segala usaha dan kebaikan yang diberikan pada saya. Tugas akhir ini saya persembahkan kepada ibu, bapak dan eyang saya sebagai wujud cinta dan terima kasih saya kepada beliau
2. Kakak Galuh Prestisa dan adik Najwa Afiani Bilqis yang selalu mendukung, mendoakan dan menyayangi saya. Terimakasih atas segala perhatian dan dukungannya serta semangatnya. Walaupun kita punya kehidupan masing-masing tetapi kita akan pulang ke satu tempat yang sama.
3. Teman-teman duar, kontrakan bri, UMC dan masih banyak lagi yang tidak bisa saya sebutkan satu persatu terimakasih atas dukungan yang luar biasa, telah menjadi teman yang sangat berharga bagi saya dan selalu ada ketika saya berada dalam kesulitan. Terimakasih telah menemani saya selama masa perkuliahan ini.
4. Teruntuk satu orang yang tidak bisa saya sebutkan namanya, saya ucapkan terima kasih karena telah membuat saya bersemangat untuk terus belajar dan belajar dan menggapai mimpi – mimpi saya.
5. Bapak Ridho Rahmadi, S. Kom., M. Sc, Ph. D dan Bapak Rian Adam Rajagede, S. Kom., M. Cs selaku dosen pembimbing, terimakasih untuk semua nasihat, bimbingan, ilmu, dan dukungan luar biasa yang selama ini telah diberikan.
6. Jurusan Informatika Universitas Islam Indonesia yang telah mewadahi saya dalam menuntut ilmu dan sebagai wadah untuk saya mengembangkan diri.
7. Serta seluruh pihak yang tidak bisa saya sebutkan satu-persatu, terimakasih atas segala perhatian dan dukungan yang luar biasa.

HALAMAN MOTO

“Sesungguhnya sesudah kesulitan itu ada kemudahan. Maka apabila kamu telah selesai (dari suatu urusan), kerjakanlah dengan sungguh-sungguh (urusan) yang lain, dan hanya kepada Tuhanmu lah hendaknya kamu berharap”

(Q.S Al-Insyirah: 6-8)

“Hai orang-orang yang beriman, bersabarlah kamu dan kuatkanlah kesabaran mu dan tetaplah bersiap siaga (di perbatasan negerimu) dan bertakwalah kepada Allah, supaya kamu beruntung.”

(Q.S Ali-Imran :200)

“Lakukan apa yang kamu mau, lakukan apapun yang kamu bisa asal jangan sekalipun kamu menyakiti orang lain dan yang bisa menghentikan mu adalah dirimu sendiri”

(Fatih Assidhiqi)

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh

Alhamdulillah, segala puji dan syukur saya panjatkan atas kehadirat Allah SWT, yang telah memberikan Ridho dan rahmat-Nya. Sehingga penulis dapat menyelesaikan laporan tugas akhir ini yang berjudul "Pengembangan Sistem Deteksi Hunian Parkir Menggunakan Metode Convolutional Neural Network" yang merupakan sebagian kecil dari perjalanan hidup saya dan juga merupakan ibadah saya kepada Allah SWT. Laporan ini disusun untuk menyelesaikan pendidikan pada jenjang Strata 1 (S1) pada Jurusan Informatika Universitas Islam Indonesia. Penulis sadar laporan ini tidak akan dapat selesai dengan waktu yang cepat tanpa dukungan serta motivasi dari berbagai pihak. Oleh karena itu penulis tidak lupa menyampaikan rasa terima kasih yang sebesar-besarnya kepada:

1. ALLAH SWT, atas limpahan rahmat dan hidayah-Nya yang selalu hadir di setiap langkah dalam memberikan kekuatan, ketaatan, dan kemampuan saya untuk dapat menyelesaikan Tugas Akhir ini dengan lancar.
2. Kedua orang tua saya, Bapak Muanis dan Ibu Dyah Kristianingsih untuk doa, dukungan, dan rasa percaya selama ini.
3. Bapak Fathul Wahid, S.T., M.Sc., Ph.D., selaku Rektor Universitas Islam Indonesia
4. Bapak Prof. Dr. Ir. Hari Purnomo, M.T., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak Hendrik, S.T., M.Eng., selaku Ketua Jurusan Informatika Universitas Islam Indonesia.
6. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc., selaku Ketua Program Studi Informatika Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia.
7. Bapak Ridho Rahmadi, S. Kom., M. Sc, Ph. D dan Bapak Rian Adam Rajagede, S. Kom., M. Cs., selaku dosen pembimbing Tugas Akhir yang memberikan peran luar biasa secara langsung dalam penyelesaian tugas akhir ini.
8. Bapak dan Ibu dosen Program Studi Informatika, yang telah memberikan ilmu yang bermanfaat kepada penulis, semoga bapak dan ibu selalu diberikan kesehatan serta lindungan dari Allah SWT.
9. Teman – teman DUAR dan kontrakan BRI yang senantiasa selalu ada, terima kasih telah memberikan segala hal yang sangat berharga serta segala bantuan yang telah diberikan kepada saya.

10. Teman – teman PIXEL angkatan 2017, kakak dan adik angkatan yang tidak bisa saya sebutkan satu per satu.
11. Teman – teman UMC yang telah menemani saya selama masa perkuliahan ini.
12. Semua pihak yang tidak bisa saya sebutkan satu per satu, terima kasih atas semua bentuk dukungannya.

Semoga segala bantuan, dukungan, bimbingan, pengajaran, serta kasih sayang yang telah diberikan kepada penulis mendapatkan kebaikan dari Allah SWT. Penulis memohon maaf apabila selama melaksanakan Tugas Akhir terdapat kekhilafan dan kesalahan. Penulis menyadari akan keterbatasan kemampuan yang dimiliki. Semoga laporan ini dapat bermanfaat bagi semua yang membaca dan menikmatinya.

Yogyakarta, 16 April 2021

(Fatih Assidhiqi)

SARI

Kurangnya informasi mengenai area parkir merupakan masalah yang saat ini dihadapi oleh para pengendara. Peningkatan jumlah kendaraan dengan lahan parkir yang dibutuhkan seringkali tidak seimbang dimana hal ini dapat menimbulkan kemacetan di berbagai tempat, salah satunya yaitu di area parkir. Untuk mendapatkan tempat parkir, seringkali pengendara harus berkeliling ke seluruh area parkir, bahkan pengendara harus keluar dari area parkir tersebut karena tidak dapat menemukan tempat parkir untuknya. Pada penelitian ini, penulis mengembangkan sistem deteksi hunian parkir menggunakan metode *Convolutional Neural Network* yang diimplementasikan menggunakan aplikasi android, sehingga pengendara dapat menggunakannya untuk mencari informasi dimana letak tempat parkir yang kosong. Hasil deteksi gambar area parkir menggunakan metode CNN dimana nilai akurasi dan *loss* yang didapatkan terbilang sangat baik, pada penelitian ini model arsitektur terbaik ada pada modifikasi VGG16 dengan akurasi pada *training set* sebesar 99,75% dan loss sebesar 0,0090, sedangkan pada *validation set* akurasi yang didapat yaitu 99,89% dan loss sebesar 0,0045, serta pada *test set* hasil akurasinya adalah 99,09% dan loss sebesar 0,0365. Hasil dari sistem ddeteksi gambar sebuah area parkir menggunakan model yang sudah dilatih sebelumnya dapat divisualisasikan dengan baik pada aplikasi android.

Kata kunci: Convolutional Neural Network, hunian parkir, android

DAFTAR ISI

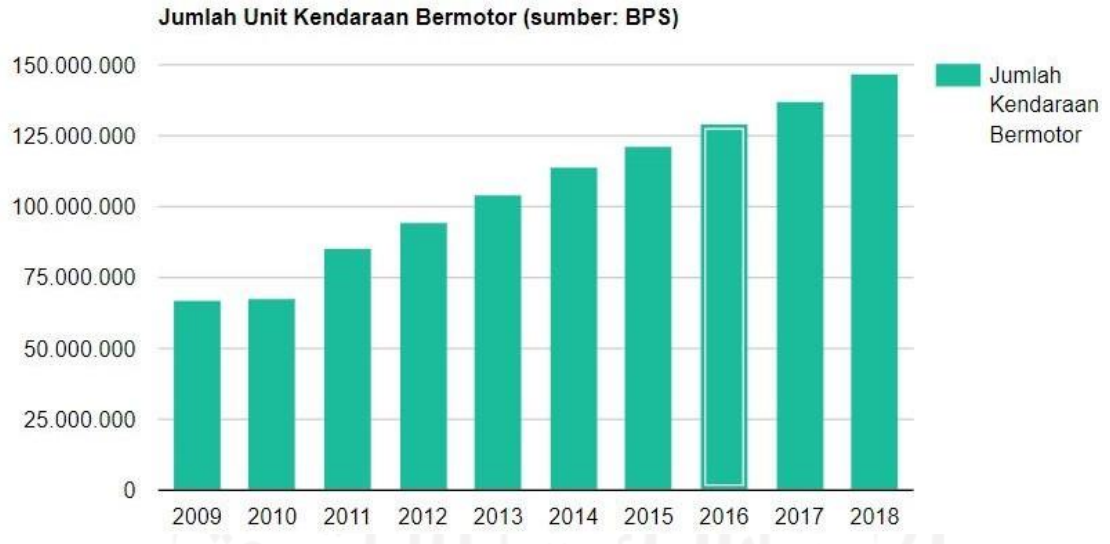
DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Area parkir saat ini sangat dibutuhkan di tempat umum yang mana area parkir merupakan sebuah fasilitas umum di luar badan jalan yang dapat berupa taman parkir atau gedung parkir. Jumlah pengguna kendaraan bermotor semakin meningkat per tahunnya, dikarenakan alat transportasi memang sudah menjadi kebutuhan pokok bagi masyarakat dari berbagai kalangan (Akbar and Jura 2019). Dapat dilihat pada grafik Gambar 1.1 dimana sejak tahun 2009 sampai dengan tahun 2018 jumlah kendaraan selalu mengalami peningkatan, dimana kenaikan jumlah kendaraan ini menimbulkan kemacetan di berbagai tempat, salah satunya di area parkir baik itu di pusat perbelanjaan maupun tempat wisata terjadi karena tidak seimbangnya jumlah kendaraan yang ada dengan jumlah tempat parkir yang dibutuhkan (Setiawan et al. 2017).



Gambar 1.1. Peningkatan jumlah kendaraan bermotor per tahun

Sumber: (B. P. Statistik 2018)

Sulitnya mencari tempat parkir yang kosong merupakan salah satu masalah yang disebabkan oleh padatnya jumlah kendaraan yang terus meningkat. Akibatnya, terjadi antrian panjang yang bahkan sampai mengganggu arus kendaraan di jalan raya (Zulkarnain and Julian 2017). Untuk mendapatkan tempat parkir seringkali pengendara harus berkeliling ke seluruh tempat parkir yang ada, bahkan pengendara harus keluar dari area parkir karena tidak

menemukan tempat parkir untuknya (K and Amini 2016). Kurangnya informasi mengenai area parkir menjadi sebuah masalah yang saat ini dihadapi yang mana dapat mengganggu kenyamanan pengendara baik wisatawan ataupun *customers* dari sebuah pusat perbelanjaan. Deteksi hunian parkir merupakan salah satu solusi untuk mengetahui bagaimana kondisi dari sebuah area parkir. Beberapa area parkir seperti mall atau area perkantoran menggunakan alat sensor untuk menentukan status suatu hunian parkir. Dalam proses instalasi dan pemeliharaan sensor dibutuhkan biaya yang mungkin tinggi khususnya di area parkir yang memuat banyak hunian parkir, hal ini dikarenakan setiap hunian parkirnya dibutuhkan satu sensor (Tanuwijaya and Fatichah 2020a) (Amato et al. 2016).

Pada penelitian sebelumnya (Nugraha, Jati, and Ahmad 2016) terdapat metode untuk mendeteksi hunian parkir dengan menggunakan pengolahan citra, yaitu *Histogram of Oriented Gradient* (HOG). HOG merupakan teknik yang menghitung nilai *gradient* dalam daerah tertentu pada suatu gambar. Tiap gambar mempunyai karakteristik yang ditunjukkan oleh distribusi *gradient,* yang diperoleh dengan membagi gambar ke dalam daerah kecil yang disebut *cell.* Tiap *cell* disusun sebuah histogram dari sebuah gradient. Kombinasi dari histogram ini dijadikan sebagai descriptor yang mewakili sebuah objek. Dalam HOG jarak kamera dengan objek sangatlah menentukan hasil deteksi hunian parkir. Semakin dekat jarak kamera dengan hunian parkir maka semakin tinggi akurasi yang didapat, tetapi cakupan slot parkir semakin sedikit, begitu juga sebaliknya, semakin jauh jarak kamera dengan hunian parkir maka akurasi semakin kecil bahkan bisa mencapai 0% tetapi cakupan slot parkir semakin banyak.

Metode lain untuk mendeteksi hunian parkir dengan mengandalkan pengolahan citra yaitu *Convolutional Neural Network* (CNN). CNN terdiri dari sejumlah lapisan tersembunyi dimana masing – masing melakukan perhitungan matematika dari input dan menghasilkan output untuk setiap lapisannya. Untuk melakukan deteksi dibutuhkan proses pelatihan pada model CNN menggunakan data *training* (Amato et al. 2016). Data yang dibutuhkan untuk pelatihan model CNN adalah data gambar dari sebuah area parkir. Terdapat beberapa tantangan dalam pengambilan gambar area parkir seperti kondisi cahaya atau bayangan yang berbeda beda pada saat pagi, siang, sore maupun malam, selain itu, terkadang terdapat benda – benda yang menghalangi kamera untuk mengambil gambar objek yang diinginkan. CNN yang dilatih sebelumnya oleh sekumpulan data citra menghasilkan kinerja yang mampu menutupi kekurangan pada metode deteksi hunian parkir sebelumnya (Acharya, Yan, and Khoshelham

2018). CNN sendiri memiliki model arsitektur yang beragam, diantaranya yaitu VGG16, AlexNet, LeNet-5, ResNet50, GoogleNet, dan lain sebagainya

Untuk memantau hasil deteksi hunian parkir dibutuhkan suatu perangkat yang dapat digunakan oleh orang banyak. Android merupakan salah satu sistem operasi yang dirancang oleh Google. Android biasa digunakan pada perangkat dengan layar sentuh seperti *smartphone* dan tablet. Pada tahun 2017, *Market share Android* di seluruh dunia menyentuh angka 73%, dengan iOS sebagai pesaing terdekatnya 19,99%. Di Indonesia sendiri 83,99% *market share* dikuasi oleh Android pada bulan Juli 2017 (Zahid 2018). Dari 265 juta total penduduk di Indonesia, 130 juta merupakan pengguna aktif media sosial. 177 juta pengguna *handphone* di Indonesia dengan 120 juta aktif di media sosial (Samsiana et al. 2020). Penggunaan Android yang sangat masif dalam kehidupan sehari – hari dapat dimanfaatkan untuk mengembangkan sistem deteksi hunian parkir pada penelitian ini.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

a. Bagaimana membangun sistem deteksi hunian parkir menggunakan *Convolutional Neural Network.*

b. Manakah model arsitektur (AlexNet, LiteAlexNet, LeNet-5, VGG16, ResNet50, GoogleNet, dan Modifikasi VGG16) yang memiliki akurasi dan *loss* terbaik dalam mendeteksi status hunian parkir.

c. Bagaimana membangun sistem untuk memantau hasil deteksi hunian parkir menggunakan aplikasi Android.

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

a. Sistem dapat mendeteksi hunian parkir menggunakan metode *Convolutional Neural Network*

b. Menemukan model arsitektur mana yang memiliki akurasi dan *loss* terbaik dalam mendeteksi status hunian parkir.

c. Aplikasi Android dapat digunakan untuk memantau hasil deteksi hunian parkir .

1.4 Manfaat Penelitian

Manfaat dari penelitian ini antara lain:

a. Dapat membantu penyelesaian masalah parkir menggunakan metode *Convolutional Neural Network.*

b. Dengan diketahuinya hasil deteksi hunian parkir diharapkan dapat membantu menyelesaikan masalah mengenai kurangnya informasi yang didapat pengendara ketika hendak parkir.

c. Hasil penelitian ini dapat dijadikan acuan untuk penelitian lebih lanjut.

1.5 Batasan Masalah

Batasan penelitian agar sesuai dengan yang dimaksudkan dan lebih terarah adalah sebagai berikut:

a. *Software* yang digunakan untuk mendeteksi hunian parkir adalah Python dengan *framework* TensorFlow.

b. Sistem pada penelitian ini berupa simulasi untuk deteksi status hunian parkir.

c. Data yang digunakan dalam penelitian ini adalah gambar area parkir *outdoor* dengan dua status hunian yaitu parkir terisi atau kosong.

d. *Dataset* diambil dari internet yaitu area parkir *cnrpark.it.*

e. Data yang digunakan memiliki 8 posisi kamera yang berbeda yaitu kamera A, kamera B, kamera C, kamera D, kamera E, kamera F, kamera G, dan kamera H.

f. Terdapat *Overlapping* pada sebagian slot parkir antar kamera.

g. Metode yang digunakan untuk menentukan status hunian parkir adalah *Convolutional Neural Network.*

h. Model arsitektur yang dibandingkan yaitu AlexNet, LiteAlexNet, LeNet-5, VGG16, ResNet50, GoogleNet, dan Modifikasi VGG16.

i. *Software* yang digunakan untuk memantau hasil deteksi hunian parkir adalah Android studio dengan Bahasa pemrograman Java.

j. *Platform* yang digunakan sebagai database adalah Firebase dengan fitur *real-time database* untuk menyimpan hasil deteksi.

1.6 Sistematika Penelitian

Sistematika penulisan memuat tentang metode penulisan yang digunakan dalam penelitian. Berikut merupakan sistematika penulisan penelitian yang terbagi menjadi lima bab

BAB I PENDAHULUAN

Berisi latar belakang masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, Batasan masalah, dan sistematika penulisan.

BAB II LANDASAN TEORI

Berisi kajian penelitian sebelumnya dan teori – teori yang digunakan dalam penelitian yang diambil dari berbagai sumber terutama penelitian – penelitian yang pernah dilakukan.

BAB III METODE PENELITIAN

Berisi tentang uraian langkah – langkah yang dilakukan untuk menyelesaikan penelitian seperti pengumpulan data, pemilihan model arsitektur terbaik, perancangan sistem, dan pengujian.

BAB IV HASIL DAN PEMBAHASAN

Berisi tentang pemaparan hasil implementasi dan hasil pengujian berdasarkan urutan langkah – langkah serta scenario pengujian pada BAB III

BAB V KESIMPULAN DAN SARAN

Berisi tentang kesimpulan dari hasil penelitian dan saran untuk penelitian selanjutnya.

BAB II
LANDASAN TEORI

21 Kajian Penelitian Sebelumnya

Penelitian mengenai deteksi hunian parkir dengan berbagai metode telah banyak dilakukan oleh para peneliti, diantaranya yaitu metode sensor, *Histogram of Oriented Gradient* (HOG), *Convolutional Neural Network* (Amato et al. 2016; Dewisita, Nuryanto, and Burhanuddin 2019; Nugraha et al. 2016). Metode sensor merupakan metode yang paling sering digunakan di area parkir yang tertutup seperti gedung pusat pembelanjaan, mall, bahkan perkantoran. Metode ini mengandalkan sensor yang dipasang di setiap hunian parkir untuk mendeteksi ada atau tidaknya mobil ditempat tersebut, kemampuan sensor untuk mendeteksi bisa mencapai 100% tetapi diperlukan sensor yang sangat banyak serta maintenance untuk setiap sensor dan ini memerlukan biaya yang tinggi (Acharya et al. 2018)

Histogram of Oriented Gradient (HOG) merupakan metode yang digunakan dalam *image processing* untuk tujuan mendeteksi objek. Pada penelitian (Nugraha et al. 2016), HOG digunakan untuk mendeteksi hunian parkir dengan menghitung nilai gradient dalam daerah tertentu pada suatu gambar. Distribusi *gradient* menunjukan karakteristik dari sebuah gambar, karakteristik tersebut diperoleh dengan mengkombinasikan daerah - daerah kecil pada gambar atau biasa disebut cell yang disusun sebuah histogram dari sebuah gradient. Metode HOG ini lebih hemat biaya dibandingkan dengan metode sensor karena fungsi pengawasan pada area parkir bisa dilakukan secara bersamaan (Acharya et al. 2018). Dalam mendeteksi suatu objek akurasi bisa mencapai 93.33% tetapi untuk mendapatkan akurasi tersebut diperlukan jarak yang dekat antara kamera dengan hunian parkir. Jika terlalu jauh akurasi yang didapat bisa mencapai 0% (Nugraha et al. 2016). Menurut (Acharya et al. 2018) metode HOG juga mempunyai kelemahan, yaitu kemampuan yang terbatas untuk beradaptasi dengan variasi objek kendaraan yang *non-linear*, variasi waktu, dan kompleks.

Pada penelitian (Alamsyah 2017) menggunakan dua metode untuk kasus pengenalan mobil, yaitu HOG dan *Support Vector Machine* (SVM). HOG digunakan sebagai fitur ekstraksi dari sebuah citra untuk mengenali ciri mobil dan ciri bukan mobil, hasil dari fitur ekstraksi diklasifikasi menggunakan metode SVM. Hasil dari penelitian ini mendapatkan akurasi terbaik yaitu 82,5% dengan pembagian data citra 1:1. Pelatihan yang menggunakan data citra dengan

pembagian ratio yang tidak seimbang akan terjadi *overfitting* bahkan penulis menyatakan bahwa belum ditemukan model yang tepat dari data latih yang diberikan.

Pada penelitian (Maulana, Fitriyah, and Prakasa 2018) yang berjudul "*Implementasi Sistem Deteksi Slot Parkir Mobil Menggunakan Metode Morfologi dan Background Subtraction*" mengandalkan garis pada area parkir sebagai pembatas antar hunian parkir menggunakan *Background Subtraction*, garis pembatas tersebut dijadikan sebuah acuan untuk mendeteksi status hunian. Hasil akurasi yang didapatkan sangat baik yaitu 100%, tetapi untuk mendapatkan akurasi tersebut dibutuhkan data gambar area parkir yang memiliki garis putih sebagai pembatas antar hunian parkir dan pada penelitian ini, data gambar area parkir hanya menampung sedikit hunian parkir.

Convolutional Neural Network (CNN) merupakan salah satu metode untuk berbagai tugas yang kompleks termasuk klasifikasi gambar, deteksi objek, segmentasi semantic. CNN terdiri dari beberapa lapisan neuron yang dapat mempelajari fitur tingkat tinggi secara efisien dari sejumlah besar data pelatihan berlabel (Li and Shen 2016). Untuk kasus pada penelitian ini dibutuhkan gambar area parkir, dimana untuk setiap hunian parkirnya diberikan label untuk dilakukan pelatihan CNN. Menurut (Liu and Wang 2017) kekuatan dari CNN tidak hanya mempelajari bobot dari fitur tetapi juga menampilkan fitur itu sendiri. CNN yang telah dilatih sebelumnya oleh data training menghasilkan kinerja yang mampu menutupi kekurangan pada metode pendeteksi hunian parkir sebelumnya dengan akurasi yang dihasilkan mencapai 99% dan CNN mampu untuk mengatasi Batasan pada metode HOG dengan mempelajari fitur yang mengidentifikasi gambar secara optimal (Acharya et al. 2018).

Pada penelitian ini akan menggunakan metode *Convolutional Neural Network* karena metode ini mempelajari fitur yang memiliki informasi spasial, yang membuat metode ini cocok untuk citra, serta metode ini mampu mengatasi kekurangan dari metode sebelumnya dimana CNN mampu untuk mengenali sebuah gambar yang kompleks dengan pembelajaran mendalam. Metode CNN dapat mendeteksi berbagai hunian parkir secara bersamaan tanpa bergantung dengan jarak kamera dan CNN mampu mengatasi Batasan pada metode HOG untuk mempelajari objek pada gambar secara optimal.

Tabel 2.1. Perbandingan Pustaka

No.	Peneliti	Metode	Data Penelitian	Hasil

1.	(Nugraha et al. 2016)	Histogram of Oriented Gradient (HOG)	Data berupa gambar area parkir tetapi tidak dituliskan bagaimana bentuk dan jenis gambarnya	HOG dapat mengidentifikasi slot parkir dengan *distance* optimal yang digunakan adalah sebesar 230,33, jika jarak lebih dari 670,75 maka slot parkir tidak dapat diidentifikasi
2.	(Alamsyah 2017)	Histogram of Oriented Gradient (HOG) dan Support Vector Machine (SVM)	Data berupa gambar mobil tampak samping	HOG-SVM dapat mengidentifikasi objek mobil, fokus dalam penelitian ini yaitu pembagian data set dengan hasil terbaik yaitu menggunakan ratio 1:1 untuk citra (+) dan citra (-) yaitu sebesar 82,5%.
3.	(Maulana et al. 2018)	Morfologi dan Background subtraction	Data berupa gambar area parkir dengan garis pembatas berwarna putih	Metode morfologi dan background subtraction dapat diterapkan dengan baik ke dalam sistem deteksi parkir. Tingkat akurasi yang didapatkan yaitu 100%, namun untuk dapat mengidentifikasi slot parkir dibutuhkan tempat parkir dengan pembatas berwarna putih
4.	(Amato et al. 2016)	Convolutional Neural Network (CNN)	Data yang digunakan sama dengan penelitian ini yaitu area parkir CNRPark, namun hanya menggunakan dua posisi kamera	CNN secara efektif dapat diterapkan untuk mendeteksi status hunian parkir dengan secara khusus menunjukan akurasi yang sangat tinggi. Pada penelitian ini juga membandingkan dua arsitektur CNN yaitu mLeNet-5 dan mAlexNet dengan beberapa pengujian dengan hasil terbaiknya yaitu 99,6%.

5.	(Tanuwijaya and Fatichah 2020b)	Convolutional Neural Network (CNN) dengan modifikasi model arsitektur AlexNet serta YOLO V3	Data yang digunakan sama dengan penelitian ini yaitu area parkir CNRPark, namun hanya menggunakan satu posisi kamera	Untuk mengidentifikasi ketersediaan tempat parkir peneliti mengusulkan menggunakan YOLO V3 dan menggunakan model arsitektur baru yaitu Lite AlexNet untuk klasifikasi tempat parkir tersedia atau ditempati dengan hasil terbaik 96,67%.
6.	(Zhang et al. 2018)	Deep Convolutional Neural Network (DCNN)	Gambar dari kamera yang dipasang di sekitar mobil (*wide angle*) untuk mengambil gambar area di sekitar mobil tersebut	*Vision-based* sistem deteksi parkir menggunakan metode DCNN dapat dengan baik mengidentifikasi area parkir di sekitar mobil yang dipasang kamera *wide angle* dengan tingkat akurasi sebesar 98,89%
7.	(Paidi, V., Fleyeh, H., Nyberg 2018)	Convolutional Neural Network (CNN)	Gambar area parkir dengan kamera pendeteksi suhu	CNN dapat mengidentifikasi slot parkir dengan baik menggunakan kamera pendeteksi suhu, peneliti juga membandingkan beberapa model arsitektur dengan hasil akurasi terbaik yaitu 96,16% denga model arsitektur ResNet18

22 Citra Digital

Citra merupakan representasi dari suatu benda atau objek. Citra terdiri dari kombinasi antara titik, garis, bidang dan warna. Secara sistematis, sebuah citra dapat didefinisikan sebagai fungsi kontinu dari intensitas cahaya pada suatu bidang dua dimensi yang disimbolkan dengan $f(x, y)$ berukuran M baris dan N kolom, dengan koordinat spasialnya adalah x dan y, dan amplitudo f di titik koordinat (x, y) yang dinamakan intensitas atau tingkat keabuan dari sebuah citra pada titik tersebut (Limbong 2020).

Pengolahan citra menggunakan komputer dapat dilakukan dengan konversi citra analog ke citra digital, proses ini dinamakan dengan digitalisasi citra atau proses representasi dari suatu citra (kontinu) secara numerik dengan nilai-nilai diskrit.

23 Citra Warna RGB (Red, Green, Blue)

Persepsi visual citra berwarna umumnya lebih kaya dibandingkan dengan citra hitam putih, dimana citra warna menampilkan warna objek seperti warna aslinya, meskipun tidak selalu tepat. Citra RGB juga disebut sebagai citra True Color. Dalam citra RGB setiap pixel memiliki 3 komponen warna, yaitu merah (R), hijau (G), dan biru (Blue). Dari setiap komponen warna citra RBG memiliki jangkauan nilai antara 0 sampai 255 (8 bit). Hal ini akan memberikan kemungkinan total warna sebanyak 255^3 atau sama dengan 16.581.375. Total ukuran bit pada citra RGB untuk setiap piksel adalah 24 bit (8-bit R, 8-bit G, 8-bit B).

24 Artificial Intelligence (AI)

Artificial Intelligence atau bisa disingkat AI merupakan sebuah kecerdasan yang ditambahkan ke dalam suatu sistem yang diatur dalam konteks ilmiah. Andreas Kaplan dan Michael Haenlein mendefinisikan kecerdasan buatan sebagai "kemampuan sistem untuk menafsirkan data eksternal dengan benar, untuk belajar dari data tersebut, dan menggunakan pembelajaran tersebut guna mencapai tujuan atau tugas tertentu melalui adaptasi yang fleksibel" (Siahaan et al. 2020).

Menurut (Ahmad 2017) terdapat tiga metode yang dikembangkan dari AI, antara lain yaitu:

1. *Machine Learning* (ML) atau pembelajaran mesin adalah teknik yang paling populer dikarenakan banyak digunakan untuk menirukan perilaku manusia untuk penyelesaian masalah tertentu.
2. *Fuzzy Logic* (FL), teknik yang satu ini digunakan mesin untuk mengadaptasi bagaimana makhluk hidup menyesuaikan kondisi dengan memberikan keputusan yang tidak kaku yaitu 0 atau 1.
3. *Evolutionary Computing* (EC). Teknik ini menggunakan skema evolusi dengan menggunakan jumlah individu yang banyak dan memberikan sebuah ujian untuk menyeleksi individu terbaik.

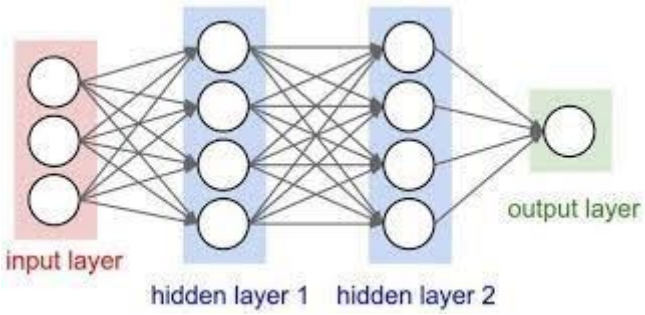
25 Deep Learning

Deep Learning merupakan cabang ilmu dari *Machine Learning* berbasis jaringan saraf tiruan untuk implementasi suatu permasalahan tertentu. Dalam *Deep Learning*, sebuah komputer dapat belajar mengklasifikasikan gambar, suara, teks atau video secara langsung. Komputer dilatih menggunakan data set berlabel dan berjumlah besar yang kemudian mengubah nilai piksel dari sebuah gambar menjadi suatu representasi internal atau *feature vector* yang digunakan untuk mendeteksi atau mengklasifikasi pola pada masukan *input* (Lecun, Bengio, and Hinton 2015).

Dalam *Machine Learning* terdapat teknik untuk menggunakan ekstraksi fitur dari sebuah data untuk klasifikasi citra atau mengenali suara. Namun, metode ini masih memiliki beberapa kekurangan dalam hal kecepatan dan akurasi. Konsep jaringan syaraf tiruan yang dalam (banyak lapisan) dapat diimplementasikan pada *Machine Learning* yang sudah ada, sehingga komputer dapat belajar dengan kecepatan, akurasi, dan skala yang besar. Prinsip tersebut berkembang hingga *Deep Learning* semakin sering digunakan pada komunitas riset dan industri untuk membantu memecahkan banyak permasalahan denga skala yang besar, seperti *Computer vision, Speech recognition,* dan *Natural Language Processing.*

Feature Engineering merupakan salah satu fitur utama dari *Deep Learning* untuk ekstraksi pola yang berguna dari sebuah data yang dapat memudahkan model untuk membedakan kelas. *Feature Engineering* juga merupakan teknik yang paling penting untuk mencapai hasil yang baik pada tugas prediksi. Namun, hal ini sulit untuk dipelajari dan dikuasai, dikarenakan kumpulan jenis data yang berbeda, maka memerlukan pendekatan teknik yang berbeda juga.

Lapisan pada *Deep Learning* terdiri atas tiga bagian yaitu *input layer, hidden layer,* dan *output layer* seperti pada Gambar 2.1. *Input layer* berisi node – node yang menyimpan sebuah nilai *input* yang tidak berubah pada fase pelatihan dan hanya bisa berubah jika diberikan nilai *input* baru. Pada *hidden layer* dapat dibuat berlapis-lapis untuk menemukan komposisi algoritma yang tepat agar dapat mengurangi nilai *error* pada *output. Output layer* berfungsi untuk menampilkan hasil perhitungan sistem oleh fungsi aktivasi pada *hidden layer* berdasarkan nilai *input.* Dengan menambahkan lebih banyak lapisan menjadikan model pembelajaran yang bisa mewakili citra berlabel dengan lebih baik.



Gambar 2.1. Layer pada Deep Learning

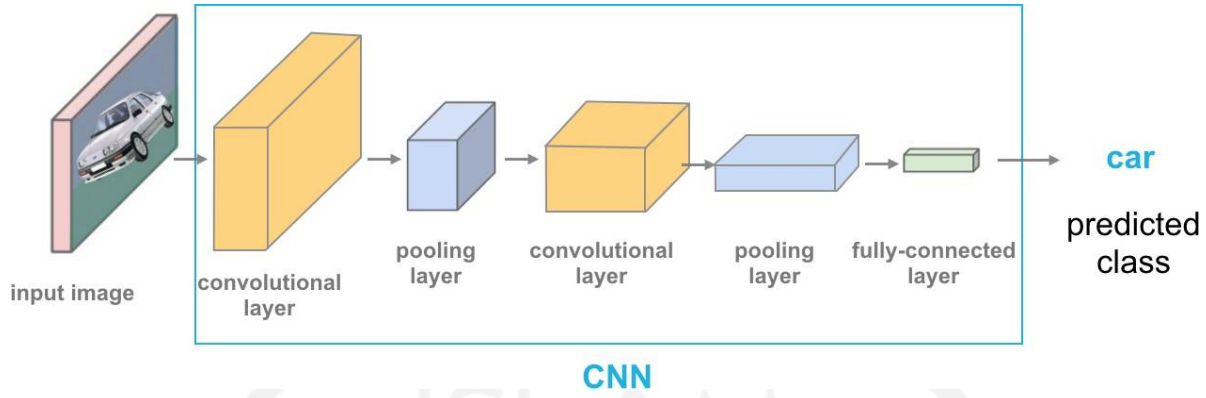
Sumber: (Eka Putra 2016)

Beberapa algoritma yang menerapkan konsep *Deep Learning* antara lain *Convolutional Neural Network* (CNN) untuk klasifikasi gambar, *Deep Belief Network – Deep Neural Network* (DBN-DNN) untuk pengenalan suara. *Recurrent Neural Network* (RNN) untuk penerjemahan bahasa, *Query-Oriented Deep Extraction* (QODE) yang berbasis *Restricted Boltzmann Machine* (RBM) untuk mendeteksi *Drug-Target Interaction* (DTI), dan *Deep Belief Network* (DBN) untuk prediksi data sesuai waktu.

2.6 Convolutional Neural Network (CNN)

Convolutional Neural network merupakan salah satu algoritma *deep learning* perkembangan dari Multilayer Perceptron (MLP) yang digunakan untuk mengolah data dua dimensi. CNN termasuk dalam jenis Deep Neural Network karena kedalaman jaringan yang tinggi dan banyak diaplikasikan pada data citra. CNN digunakan untuk mengklasifikasikan data berlabel atau menggunakan metode *supervised learning*, yang mana cara kerjanya adalah terdapat data yang dilatih dan terdapat variabel target, sehingga tujuan dari metode ini adalah mengelompokkan suatu data ke data yang sudah ada.

CNN memiliki dua metode dalam proses mengenali sebuah gambar yaitu *feedforward* sebagai tahap untuk klasifikasi/prediksi dan *backpropagation* sebagai tahap pembelajaran (Sofia 2018). CNN terdiri dari beberapa lapisan, dengan *convolutional layer*, *pooling layer* dan *fully connected* sebagai lapisan utama seperti Gambar 2.2. Setiap lapisan terbuat dari node berdasarkan data *input* dan menghasilkan *output* deteksi (Camacho 2018).

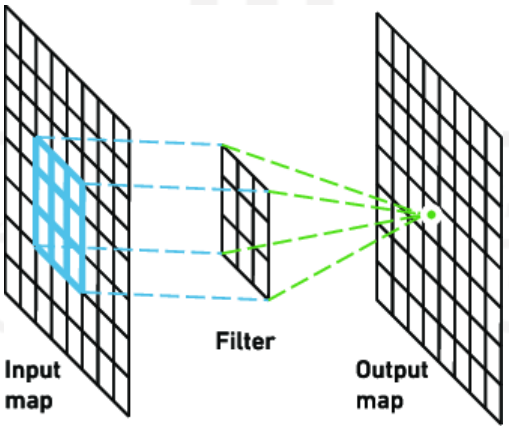


Gambar 2.2 Arsitektur sederhana CNN

Sumber: (Camacho 2018)

2.6.1 Convolutional Layer

Konvolusi memproses setiap pixel dari suatu gambar berdasarkan pada nilai pixel itu sendiri dan nilai pixel tetangganya dengan melibatkan kernel atau matrix yang mempresentasikan bobot (Ilahiyah and Nilogiri 2018). *Convolutional layer* merupakan neuron yang terdiri dari pixel – pixel suatu gambar yang disusun sedemikian rupa sehingga terbentuk sebuah filter dengan panjang dan tinggi pixel seperti pada Gambar 2.3. Filter yang terbentuk akan digeser ke seluruh bagian dari gambar dimana terjadi operasi "dot" antara input dengan nilai dari filter tersebut hingga menghasilkan output yang disebut *feature map* (Lina 2019). Operasi konvolusi seperti pada persamaan (2.1)



Gambar 2.3. Contoh *convolutional layer*

Sumber: (Yakura et al. 2018)

$$s(t) = (x * w)(t) \tag{2.1}$$

Fungsi *s(t)* memberikan output tunggal berupa *feature map.* Argumen pertama adalah *input* yang merupakan *x* dan argumen kedua *w* sebagai kernel atau filter. Apabila dilihat *input* sebagai citra dua dimensi, maka bisa dikatakan *t* sebagai piksel dan menggantinya dengan *i* dan *j*. Maka dari itu, operasi untuk konvolusi ke *input* dengan lebih dari satu dimensi dapat ditulis seperti persamaan (2.2).

$$S(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{2.2}$$

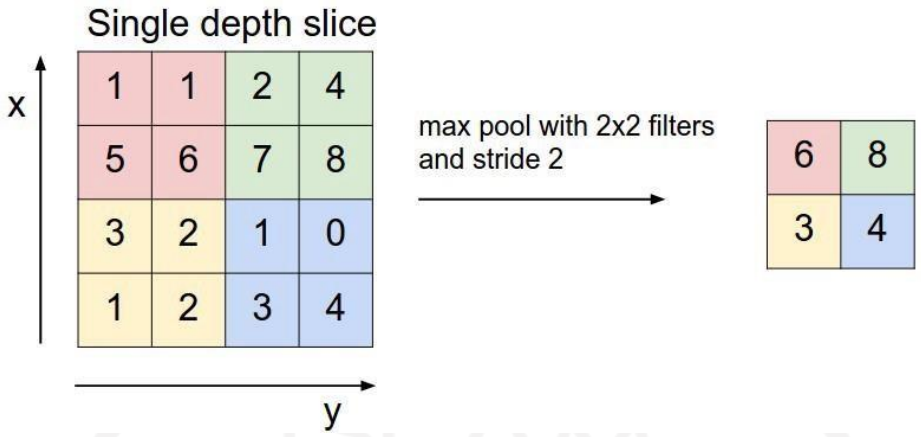
Persamaan tersebut merupakan perhitungan dasar dalam operasi konvolusi, nilai piksel yaitu *i* dan *j* dari citra. K sebagai kernel atau filter dan perhitungan tersebut bersifat kumutatif, I sebagai *input* dan kernel yang dapat dibalik relatif terhadap *input.* Penentuan volume *output* juga dapat ditentukan dari masing-masing lapisan dengan *hyperparameters*. *Hyperparameters* yang digunakan pada persamaan (2.3) digunakan untuk menghitung banyaknya neuron aktivasi dalam sekali *output*.

$$\frac{(W - F + 2P)}{S + 1} \tag{2.3}$$

Berdasarkan persamaan di atas, dapat dihitung ukuran spasial dari volume *output* yang mana *hyperparameters* yang digunakan adalah ukuran volume (W), filter atau kernel (F), *Stride* yang dipakai (S), dan jumlah *padding* nol yang digunakan (P).

2.6.2 Pooling Layer

Lapisan lain yang penting setelah *convolutional layer* adalah *pooling layer*. *Pooling layer* terdiri dari sebuah filter dengan ukuran dan stride tertentu yang bergeser ke seluruh area pada *feature map. Pooling layer* bertujuan mengurangi dimensi dari *feature map* untuk mempercepat komputasi. *Max pooling* dan *Average pooling* merupakan salah satu variasi dari *pooling layer* yang paling populer (Lina 2019). Perbedaan pada kedua *pooling layer* tersebut adalah jika kita menggunakan *max pooling* 2x2 dengan stride 2 maka pada setiap pergeseran akan dipilih nilai maximum pada area 2x2, sedangkan *Average pooling* akan memilih rata - rata (Anon n.d.).



Gambar 2.4 Contoh max-pooling dengan ukuran 2x2

Sumber: (Anon n.d.)

Penentuan volume *output* hasil *pooling layer* juga dapat dihitung menggunakan *hyperparameters* seperti persamaan (2.4). Pada *pooling layer*, *padding* tidak umum dilakukan tetapi mungkin ditemukan.

$$\frac{(W - F)}{S + 1} \tag{2.4}$$

Untuk menentukan volume *output* hasil *pooling layer* menggunakan *hyperparameter* seperti W sebagai ukuran volume filter, F sebagai filter atau kernel, S sebagai *Stride* yang dipakai.

2.6.3 ReLU (Rectified Linear Unit)

Setiap *convolutional layer* diikuti lapisan aktivasi ReLU. ReLU lebih sering digunakan karena berfungsi baik dari pada fungsi nonlinear yang lain. Dengan menggunakan fungsi aktivasi ini, jaringan dapat berlatih lebih cepat tanpa membuat perbedaan akurasi yang signifikan (Thomas and Bhatt 2018).

Pada fungsi ReLU nilai *output* dari *neuron* bisa dinyatakan 0 jika nilai *input* adalah negatif dan nilai output dari neuron sama dengan nilai *input* jika nilai *output* pada fungsi aktivasi adalah positif (Lina 2019).

$$y(\mu) = \max(0, \mu) \quad (2.5)$$

Untuk mendapatkan nilai $y(\mu)$ yaitu *output* dari fungsi ReLU dibutuhkan sebuah masukan input yaitu μ dan mencari nilai maksimal dari 0 dan μ.

2.6.4 Fully-connected layer

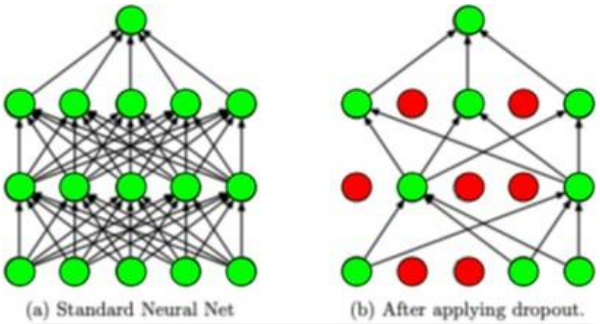
Fully-connected layer memiliki neuron dengan koneksi ke semua aktivasi di lapisan sebelumnya (Thomas and Bhatt 2018). Misalnya, untuk *fully-connected layer* yang menghasilkan tumpukan *output* dengan tinggi dan lebar 20 pixel dengan kedalaman 10 pixel maka *fully-connected layer* akan menghasilkan 20x20x10 = 4000 *inputs.* (Camacho 2018).

Perbedaan antara *fully connected layer* dan *convolution layer* biasanya adalah neuron di *convolution layer* terhubung hanya ke daerah tertentu pada *input*, sedangkan *fully connected layer* mempunyai neuron yang secara keseluruhan terhubung. Namun, kedua lapisan tersebut masih mengoperasikan produk dot, sehingga fungsinya tidak begitu berbeda.

2.6.5 Dropout Regularization

Dropout merupakan teknik regulasi jaringan syaraf dimana beberapa neuron akan dipilih secara acak dan tidak dipakai selama pelatihan. Neuron ini dapat dibuang secara acak. Kontribusi dari neuron yang dibuang diberhentikan sementara jaringan dan bobot baru juga tidak diterapkan pada neuron pada saat melakukan *backpropagation. Dropout Regularization* seperti pada Gambar 2.5.

Dropout merupakan proses mencegah terjadinya overfitting serta mempercepat proses pembelajaran. *Dropout* mengacu kepada hilangnya neuron berupa *hidden layer* maupun *visible layer* di dalam jaringan.



Gambar 2.5. Contoh implementasi Dropout Regularization
Sumber: (Mele and Altarelli 2014)

2.6.6 Softmax Classifier

Softmax Classifier merupakan bentuk lain dari algoritma *Logistic Regression* yang biasa digunakan untuk klasifikasi. Standar klasifikasi yang biasa dilakukan pada algoritma *Logistic Regression* adalah tugas untuk klasifikasi kelas biner. *Softmax Classifier* seperti pada persamaan (2.6).

$$f_i(z) = \frac{e^{z_i}}{\sum_{j=i}^{k} e^{z_j}} \qquad (2.6)$$

Notasi f_i menunjukan hasil fungsi untuk setiap elemen ke-i pada vektor keluaran kelas. Argumen z adalah hipotesis yang diberikan oleh model pelatihan agar dapat diklasifikasi oleh fungsi *Softmax* dengan *k* sebagai banyaknya elemen neuron pada lapisan *output.*

Softmax juga memberikan hasil yang lebih intuitif dan juga memiliki interpretasi probabilistic yang lebih baik dibandingkan dengan algoritma klasifikasi lainnya. *Softmax* memungkinkan untuk menghitung probabilitas untuk semua label, dari label yang ada akan diambil sebuah vektor nilai bernilai riil dan merubahnya menjadi vektor dengan nilai antara nol dan satu yang bila semua dijumlahkan bernilai satu.

2.6.7 Crossentropy Loss Function

Loss function atau *cost function* merupakan fungsi untuk menghitung nilai kerugian yang terkait dengan semua kemungkinan yang dihasilkan oleh sebuah model. *Loss function* yang baik adalah fungsi yang menghasilkan error serendah mungkin.

Crossentropy loss function atau bisa disebut juga *logarithmic loss, log loss,* atau *logistic loss.* Pada *Crossentropy,* membandingkan setiap kemungkinan kelas yang diprediksi dengan

kelas aktual atau *output* yang diinginkan 0 atau 1, nilai *loss* dihitung dengan memperbaiki probabilitas berdasarkan seberapa jauh nilai prediksi dengan nilai yang sebenarnya. *Crossentropy loss* digunakan saat menyesuaikan bobot model selama pelatihan. Tujuannya untuk meminimalisir nilai *loss*, yaitu semakin kecil *loss* semakin baik modelnya. Untuk menghitung nilai *Crossentropy* seperti pada persamaan.

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i) \tag{2.7}$$

Dengan n sebagai banyaknya kelas yang ada, dimana t_i adalah kelas aktual atau kelas sebenarnya dan p_i adalah probability *softmax* dari kelas ke i^{th}

2.6.8 Optimizer

Algoritma optimasi bertujuan untuk menemukan bobot optimal dengan meminimalkan kesalahan dan memaksimalkan akurasi. *Gradient Descent* merupakan salah satu algoritme paling populer untuk melakukan pengoptimalan *Neural Network. Gradient Descent* adalah suatu algoritma yang bertujuan meminimalkan fungsi kerugian $J(\theta)$ dengan parameter model $\theta \epsilon R^d$ dengan memperbarui parameter ke arah berlawanan dari gradien fungsi kerugian $\nabla_\theta J(\theta)$. *Learning rate* η menentukan ukuran langkah yang diambil untuk mencapai nilai lokal minimum.

2.6.8.1 Stochastic Gradient Descent (SGD)

SGD melakukan pembaruan parameter untuk setiap contoh pelatihan $x^{(i)}$ dan label $y^{(i)}$. SGD menghilangkan redundansi dengan melakukan pembaruan satu per satu. Oleh karena itu biasanya lebih cepat dan juga dapat digunakan untuk pembelajaran online. SGD sering melakukan pembaruan dengan varian tinggi yang menyebabkan fungsi kerugian sangat berfluktuasi. Fungsi optimasi SGD seperti pada persamaan (2.8).

$$\theta_{i+1} = \theta_i - \eta . \nabla_\theta J \tag{2.8}$$

θ_{i+1} merupakan nilai berat baru hasil dari fungsi optimasi SGD dengan θ_i sebagai berat sekarang. η sebagai besaran langkah atau biasa disebut dengan *learning rate,* serta $\nabla_\theta J$ sebagai nilai *error*.

2.6.8.2 RMSprop

Root Mean Square Propagation atau RMSprop adalah versi khusus Adagrad yang telah dikembangkan oleh Profesor Geoffrey Hinton. RMSProp mirip dengan Adaprop, RMSprop sendiri adalah pengoptimalan yang memanfaatkan besarnya gradien terbaru untuk menormalkan gradien yang tetap menjaga rata-rata bergerak diatas gradien *root mean square*. RMSprop juga mencoba untuk memperbaiki nilai learning rate untuk setiap iterasi. Fungsi untuk memperbaharui parameter pada RMSprop seperti pada persamaan (2.9)

$$v_t = v_{t-1} + (1 + p)\, g_t^2 \tag{2.9}$$

v_t merupakan rata - rata eksponensial pada gradien dengan g_t sebagai nilai gradien *t* selama w_J. v_t akan digunakan sebagai parameter untuk *update* nilai *learning rate* seperti pada persamaan (2.10).

$$\Delta w_t = - \frac{\eta}{\sqrt{v + {}_t \in}} * g_t \tag{2.10}$$

Δw_t merupakan perubahan nilai yang akan digunakan sebagai parameter untuk *update* berat pada persamaan (2.11). Perubahan nilai tersebut ditentukan oleh η sebagai *learning rate*, v_t sebagai rata - rata eksponensial pada gradien, ∈ sebagai nilai kesalahan pada gradient, dan g_t sebagai komponen panjang arah gradien.

$$\theta_{t+1} = \theta_t - \Delta w_t \tag{2.11}$$

θ_{t+1} merupakan nilai berat baru dengan θ_t sebagai nilai berat lama dan Δw_t sebagai besaran perubahan nilai untuk *update* berat baru.

2.6.8.3 Adaptive Moment Estimation (ADAM)

Adam merupakan metode yang menghitung kecepatan pembelajaran adaptif untuk setiap parameter. Selain menyimpan rata-rata yang menghilang secara eksponensial dari kuadrat terdahulu. Adam mempertahankan rata-rata gradien lampau yang menghilang secara eksponensial. Adam adalah salah satu optimasi yang menggabungkan metode Adagrad dan RMSProp. Fungsi untuk memperbaharui parameter pada Adam seperti pada persamaan (2.12).

$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * g_t \quad (2.12)$$

v_t merupakan *exponential average of gradients along* w_j dengan β_1 sebagai nilai *constant hyperparameter* dan g_t sebagai nilai gradien *t* selama w_J.

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2 \quad (2.13)$$

s_t merupakan *exponential average of squares gradients along* w_j dengan β_2 sebagai nilai *constant hyperparameter* dan g_t sebagai nilai gradien *t* selama w_J.

$$\Delta w_t = -\eta \frac{v_t}{\sqrt{s + \epsilon_t}} * g_t \quad (2.14)$$

Δw_t merupakan perubahan nilai yang akan digunakan sebagai parameter untuk *update* berat. Perubahan nilai tersebut ditentukan oleh η sebagai *learning rate*, v_t sebagai *exponential average of gradients*, s_t sebagai *exponential average of squares gradient*, ϵ sebagai nilai kesalahan pada gradient, dan g_t sebagai komponen panjang arah gradien.

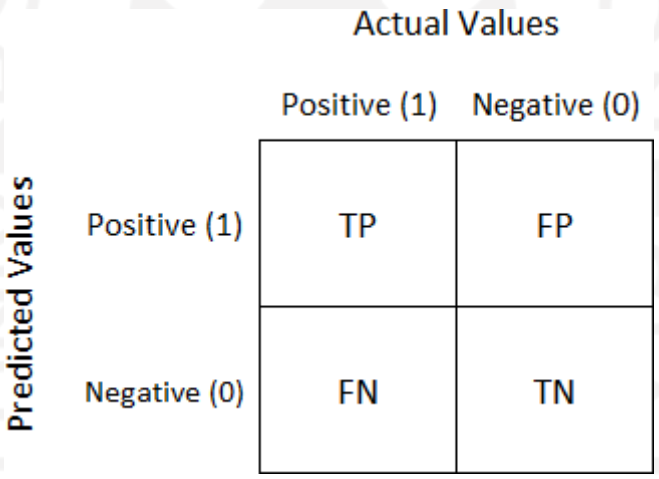
$$\theta_{t+1} = \theta_t - \Delta w_t \quad (2.15)$$

θ_{t+1} merupakan nilai berat baru dengan θ_t sebagai nilai berat lama dan Δw_t sebagai besaran perubahan nilai untuk *update* berat baru.

2.6.9 Akurasi

Akurasi merupakan salah satu metode untuk mengukur performa dari suatu model klasifikasi. Beberapa istilah umum yang digunakan dalam pengukuran kinerja model klasifikasi adalah *positive tupple* dan *negative tupple. Positive tupple* adalah *tupple* yang menjadi fokus pembahasan, sedangkan *negative tupple* adalah *tupple* selain yang sedang menjadi fokus pembahasan.

Beberapa istilah lain yang menjadi dasar untuk pencarian nilai *precision, recall,* dan akurasi adalah *true positive* (TP), *true negative* (TN), *false positive* (FP), dan *false negative* (FN). Istilah tersebut dapat ditulis sebagai suatu matriks yang disebut *confusion matrix* seperti pada Gambar 2.6.



Gambar 2.6. Confusion matrix

Nilai TP didefinisikan sebagai *positive tupple* yang diklasifikasikan dengan benar oleh model. TN adalah *negative tupple* yang diklasifikasikan dengan benar oleh model, FP adalah *negative tupple* yang diklasifikasikan sebagai kelas positif oleh model, sedangkan FN merupakan *positive tupple* yang diklasifikasikan sebagai kelas negatif oleh model klasifikasi. Dengan menggunakan nilai-nilai tersebut dapat dihitung nilai akurasi seperti pada persamaan (2.16).

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.16)$$

27 Android

Android merupakan sebuah *operating system* yang dikembangkan oleh Google berbasis Linux. Android memiliki *operating system* yang *open-source* sehingga setiap orang dapat *upload* dan *download source code* android dan menggunakannya untuk *hardware* yang dimiliki masing-masing.

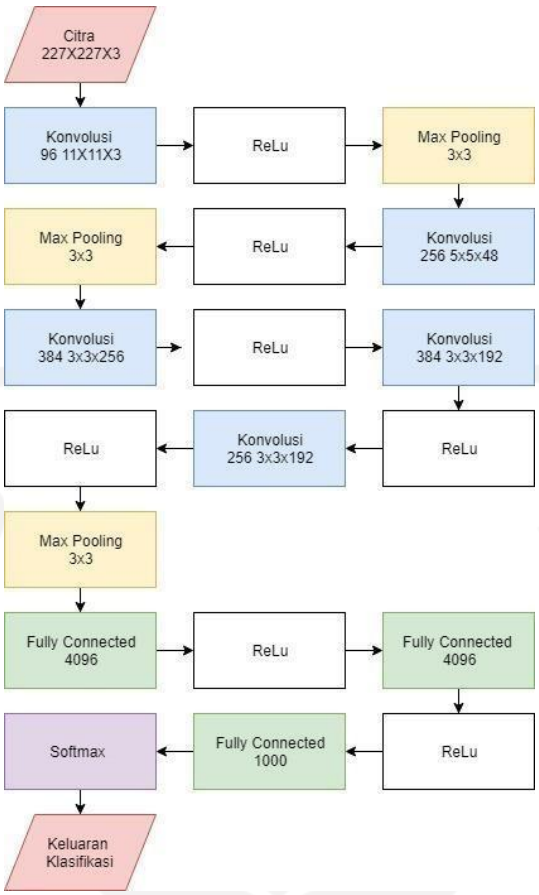
Pada penelitian ini, untuk memantau hasil deteksi hunian parkir *operating system* yang akan digunakan adalah android. Android studio merupakan sebuah *software* untuk membuat aplikasi android dengan Bahasa Pemrograman Java. Salah satu keunggulan menggunakan android studio yaitu memberikan akses ke *Android Software Development Kit* (SDK). SDK merupakan sebuah ekstensi dari kode java yang memperbolehkannya untuk berjalan mulus di *device* android.

28 Firebase

Firebase merupakan sebuah *platform* database dari *google* yang sekarang ini banyak digunakan bagi para *developer*. *Firebase* memiliki banyak SDK yang memungkinkan untuk mengintegrasikan layanan ini dengan *android, iOS, javascript, C++, Python,* hingga *unity.* Menurut (Ramadani 2017) Salah satu keunggulan dari *firebase* adalah *real-time database,* sebuah *cloud-hosted database* yang dapat menyimpan dan melakukan sinkronisasi data secara *real-time* untuk setiap *client* yang terhubung.

29 Model arsitektur AlexNet

AlexNet merupakan salah satu model arsitektur CNN yang telah dikembangkan oleh (Alex Krizhevsky, Ilya Sutskever 2007). Seperti model arsitektur lainnya, Secara garis besar *AlexNet* dibagi menjadi dua kelompok lapisan. Pertama adalah lapisan ekstraksi fitur dan yang kedua adalah lapisan klasifikasi. Masukan citra pada *AlexNet* berukuran 227x227x3 dengan proses konvolusi sebanyak lima kali menggunakan *kernel, filter, padding,* dan *stride* yang berbeda-beda. Untuk proses pooling pada arsitektur ini menggunakan *Max Pooling* atau mengambil nilai tertinggi dari matriks yang akan digunakan. Untuk lebih terperinci, model arsitektur AlexNet seperti pada Gambar 2.7.

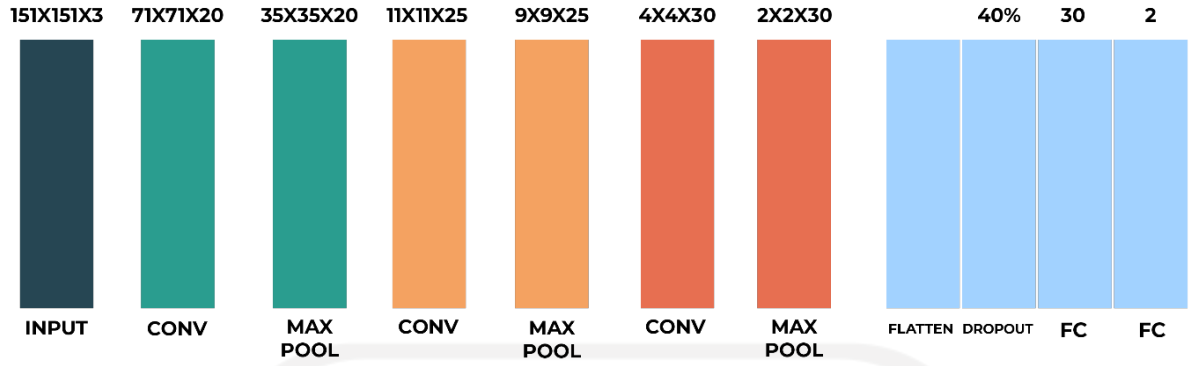


Gambar 2.7. Model Arsitektur AlexNet

Sumber: (Ilahiyah and Nilogiri 2018)

2.10 Model arsitektur LiteAlexNet

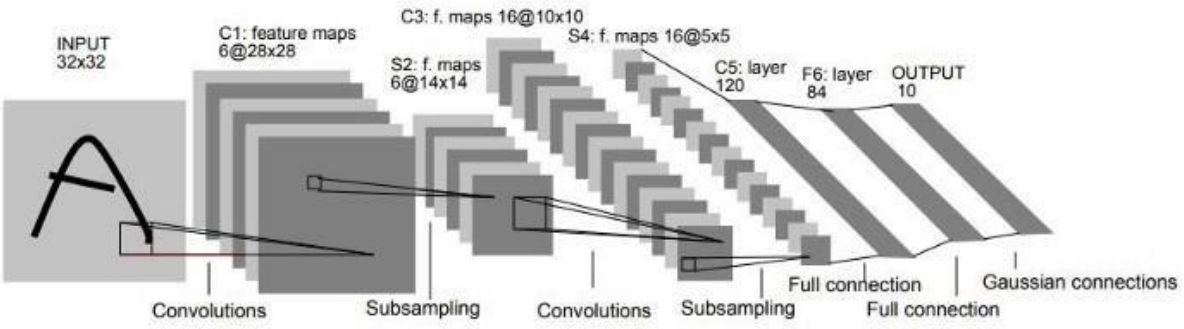
LiteAlexNet merupakan sebuah arsitektur CNN yang dikembangkan oleh Evan Tanuwijaya dan Chastine Fatichah (Tanuwijaya and Fatichah 2020a). LiteAlexNet sendiri merupakan model arsitektur yang berbasis AlexNet dengan jumlah lapisan dan parameter yang lebih kecil. Pengurangan jumlah lapisan dan parameter bertujuan untuk mempercepat proses *computing*, Alexnet dengan delapan lapisan dan Lite Alexnet dengan 5 lapisan. Pada LiteAlexNet ukuran kernel lapisan konvolusi menggunakan ukuran yang sama dengan AlexNet yaitu 11x11 *filter* 20, 5x5 *filter* 25, dan 3x3 *filter* 30. Pada *fully connected layer*, LiteAlexNet hanya menggunakan satu lapisan dengan 30 *hidden layer* dan lapisan *output* menghasilkan satu nilai. Fungsi aktivasi yang digunakan pada *convolution layers* dan *fully connected layer* adalah ReLU, sedangkan pada lapisan output menggunakan fungsi aktivasi Sigmoid. Arsitektur LiteAlexNet seperti pada Gambar 2.8.



Gambar 2.8. Model Arsitektur LiteAlexNet

2.11 Model arsitektur LeNet-5

LeNet-5 merupakan salah satu arsitektur CNN lama yang telah dikembangkan sejak tahun 1998 oleh *French-American computer scientist* Yann Andre Lecun, Leon Bottou, Yoshua Bengio, dan Patrick Haffner (Lecun et al. 2015) . LeNet-5 terdiri dari tujuh lapisan dengan dua set lapisan konvolusi, dan dua set *Average pooling layers* lalu diikuti oleh *Fully-connected layer*. Input yang dibutuhkan adalah 28x28 dengan lapisan pertama memiliki kernel berukuran (5,5) dan filter 6. Jumlah neuron pada model arsitektur LeNet-5 cenderung sedikit yaitu 44.426. Model arsitektur LeNet-5 seperti pada Gambar 2.9

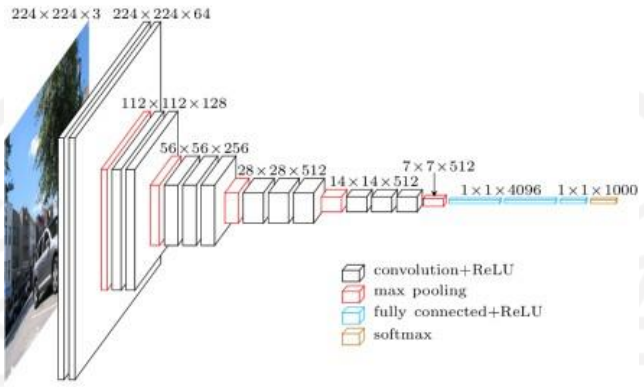


Gambar 2.9. Model arsitektur LeNet-5

2.12 Model arsitektur VGG16

VGG16 merupakan salah satu model arsitektur CNN yang digunakan untuk memenangkan kompetisi ILSVR sebagai *runner-up* pada tahun 2014. VGG16 dikembangkan oleh Xiangyu Zhang, Jianhua Zou, Kaiming He, dan Jian Sun (Zhang et al. 2016). VGG16 dianggap sebagai salah satu model arsitektur yang sanggat baik hingga saat ini (Thakur 2019). Model Arsitektur ini memiliki banyak *hyper-parameter* dengan 3x3 *convolution layer* dengan

stride 1 dan selalu menggunakan *padding* dan *maxpool layer* yang sama yaitu 2x2 dengan *stride* 2. Pada tahap akhir VGG16 memiliki 2 *fully connected layer* yang diikuti *softmax* untuk *output*. Model arsitektur ini memiliki jaringan yang cukup besar dan memiliki sekitar 138 juta parameter. Arsitektur VGG16 seperti pada Gambar 2.10.



Gambar 2.10. Model arsitektur VGG16

2.13 Model arsitektur GoogleNet

GoogleNet merupakan model arsitektur pemenang kompetisi ILSVRC 2014 yang telah dikembangkan oleh (Szegedy et al. 2015). Arsitektur GoogleNet terdiri dari 22 lapisan termasuk 27 *pooling layers*. Salah satu metode GoogleNet untuk mencapai efisiensi adalah dengan pengurangan gambar masukan sekaligus mempertahankan informasi penting. Arsitektur GoogleNet seperti pada Gambar 2.11.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Gambar 2.11. Model arsitektur GoogleNet

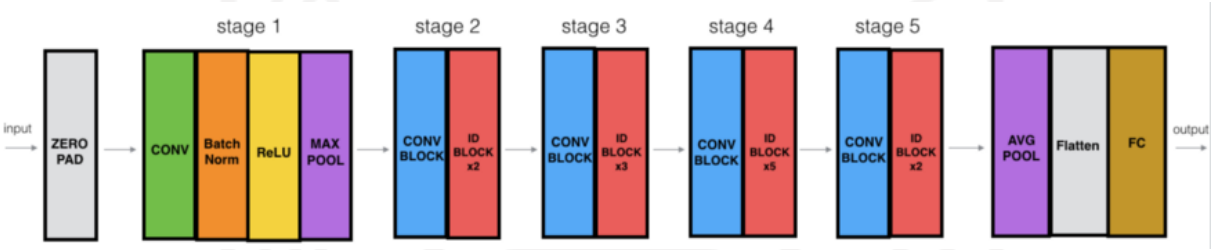
Arsitektur GoogleNet memiliki sembilan *inception module* yang di dalamnya terdapat dua *max-pooling layer* dengan tujuan untuk menurunkan jumlah sampel *input* pada saat proses *fed forward.* Pada arsitektur GoogleNet juga terdapat komponen yang diterapkan untuk mencegah *overfitting,* komponen tersebut dikenal sebagai *auxiliary classifier*. *Auxiliary classifier* terdiri dari *average pool layer,* lapisan konvolusi, dua *fully connected layer, dropout layer,* dan terakhir lapisan liner dengan fungsi aktivasi *softmax.* Setiap *auxiliary classifier* menerima *input* dari *inception module. Auxiliary classifier* hanya digunakan selama proses *training* dan dihapus selama inversi, tujuan dari *auxiliary classifier* adalah untuk melakukan klasifikasi berdasarkan *input* dalam bagian tengah jaringan dan menambahkan nilai *loss* yang dihitung selama proses *training* kembali ke nilai *loss* total.

2.14 Model arsitektur ResNet50

Residual Networks (ResNet) adalah jaringan saraf klasik yang digunakan sebagai tulang punggung untuk banyak tugas visi komputer. Terobosan mendasar ResNet adalah memungkinkan melatih jaringan neural yang sangat dalam dengan lebih dari 150 lapisan. ResNet50 Merupakan versi yang lebih kecil dari ResNet152. ResNet sendiri telah dikembangkan oleh Kaiming He, Xiangyu Zhang, Shaoqing Ren, dan Jian Sun (He et al. 2016).

Kekuatan ResNet yaitu pertama kali memperkenalkan konsep *skip connection*. *Skip Connection* berfungsi untuk mengurangi masalah hilangnya gradien dengan memungkinkan alternatif jalan pintas untuk mengalir melewati gradien dan memungkinkan model untuk mempelajari fungsi identitas yang memastikan bahwa lapisan yang lebih tinggi akan melakukan setidaknya sebaik lapisan di bawahnya.

ResNet50 terdiri dari lima *stages* dengan proses konvolusi dan *identity block.* Setiap bagian konvolusi memiliki tiga bagian *convolution layers* dan pada *identity block* juga memiliki tiga *convolution layer.* Model arsitektur ResNet50 seperti pada Gambar 2.12.



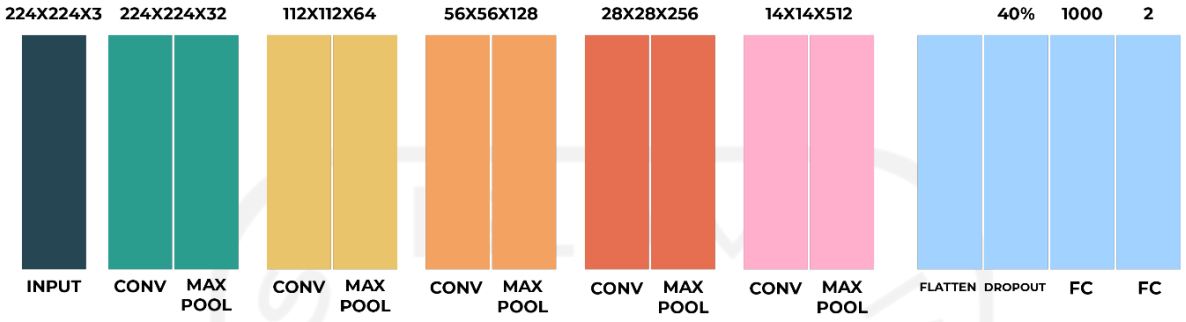
Gambar 2.12. Model arsitektur ResNet50

2.15 Model arsitektur Modifikasi VGG16

Modifikasi VGG16 merupakan sebuah model arsitektur untuk eksperimen pada penelitian ini. Alasan memilih model arsitektur VGG16 dikarenakan VGG16 memiliki tingkat akurasi yang sangat baik yaitu bisa mencapai 99% seperti pada penelitian (Tanuwijaya and Fatichah 2020a), tetapi dalam proses *computing*, VGG16 memiliki waktu yang cukup besar dikarenakan penggunaan jumlah layer yang banyak. Pada dasarnya modifikasi VGG16 sama seperti LiteAlexNet, dimana LiteAlexNet merupakan hasil modifikasi dari model arsitektur AlexNet dengan mengurangi jumlah lapisan dan parameter yang bertujuan untuk mempercepat proses *computing*.

Modifikasi VGG16 yang merupakan model arsitektur berbasis VGG16 dengan mengurangi jumlah lapisan dan hidden layer pada *fully connected layer,* serta menambahkan *dropout*. Pada VGG16 terdapat 16 lapisan dengan 13 lapisan konvolusi dan 3 lapisan *fully connected layers,* sedangkan pada modifikasi VGG16 terdapat 7 lapisan dengan 5 lapisan konvolusi dan 2 *fully connected layers*, pengurangan jumlah lapisan ini bertujuan untuk mempercepat proses *computing.* Parameter yang digunakan pada VGG16 tidak jauh berbeda, dimana ukuran kernel, *stride*, *padding*, dan fungsi aktivasi yang digunakan tetap sama, sedangkan *filters* yang digunakan yaitu 32, 64, 128, 256,dan 512. Pada *pooling layer* tetap

menggunakan Max Pooling layers. *Dropout* yang ditambahkan sebanyak 40% dengan *hidden layer* pada *fully connected layer* yaitu 1000. Model arsitektur modifikasi VGG16 seperti pada Gambar 2.13.



Gambar 2.13. Model arsitektur Modifikasi VGG16

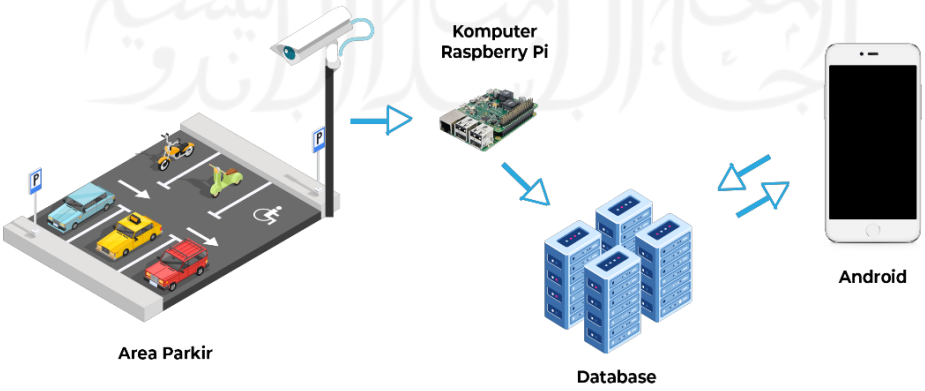
BAB III
METODOLOGI PENELITIAN

3.1 Konsep penelitian

Dalam penelitian ini terjadi beberapa perubahan konsep, dikarenakan faktor pandemi Covid 19 yang menjadikan beberapa kebijakan muncul seperti kegiatan belajar dilakukan secara daring, *social distancing*, pembatasan kegiatan di luar rumah, dan lain sebagainya, sehingga beberapa rencana awal tidak dapat terlaksana dengan semestinya. Berikut merupakan konsep awal dan konsep akhir pada penelitian ini.

3.1.1 Konsep awal

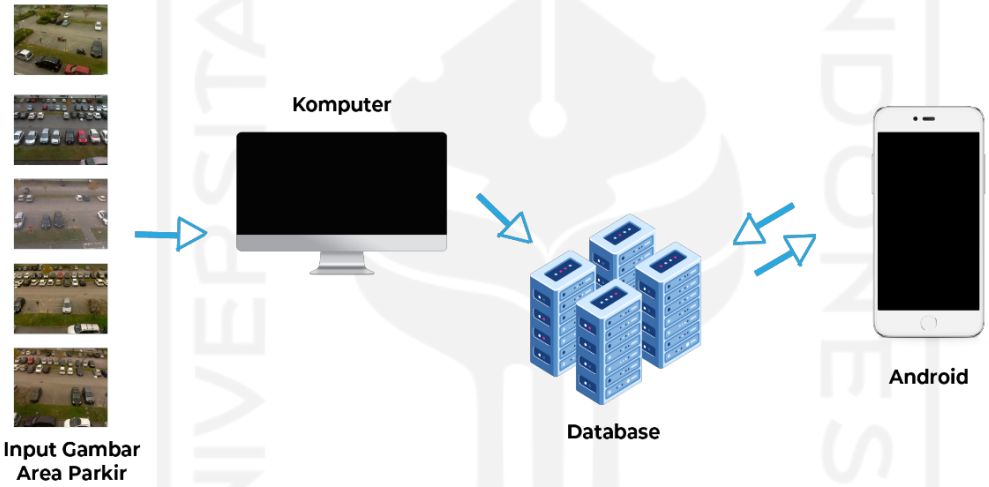
Pada konsep awal ini, peneliti berencana untuk memasang kamera CCTV pada suatu area parkir di Universitas Islam Indonesia untuk mendapatkan data yang nantinya akan berguna untuk proses deteksi status hunian parkir. Serta dengan menggunakan kamera CCTV tersebut yang telah dipasangkan komputer *mini* raspberry pi, dimana komputer raspberry akan diinstall sistem deteksi gambar yang telah dilatih sebelumnya. Hasil deteksi komputer *mini* tersebut akan dikirimkan ke database. Peneliti juga berencana untuk membuat aplikasi android yang nantinya dapat *request* data hasil deteksi yang dikirimkan oleh komputer *mini* dan menampilkan hasil deteksi tersebut pada aplikasi android secara *real-time*. Tetapi dikarenakan faktor pandemi Covid 19 yang mengharuskan beberapa kegiatan dilaksanakan di rumah dan kebijakan *social distancing*, sehingga konsep awal ini tidak dapat terlaksana dengan semestinya. Ilustrasi konsep awal pada penelitian ini seperti Gambar 3.1



Gambar 3.1. Konsep Awal

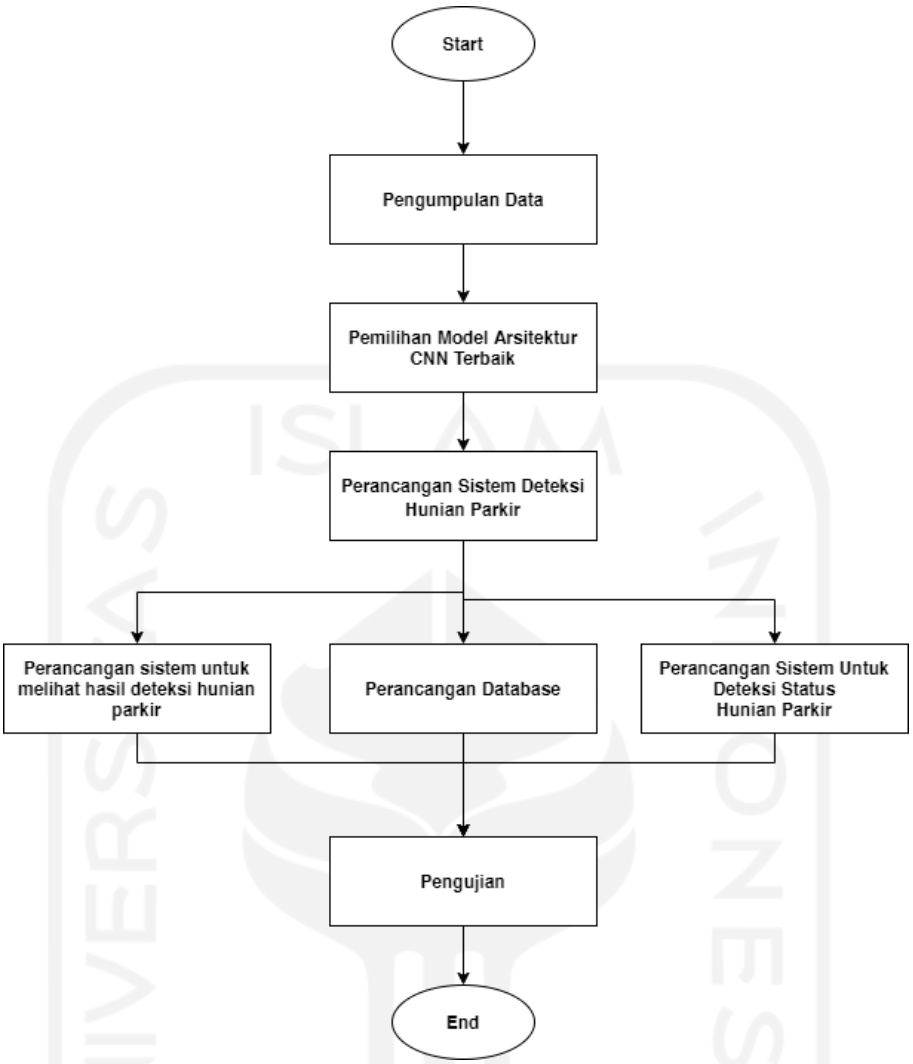
3.1.2 Konsep akhir

Dikarenakan beberapa faktor yang telah dijelaskan sebelumnya, terdapat perubahan realisasi konsep awal penelitian, hingga konsep akhir pada penelitian ini adalah hanya sebatas simulasi untuk deteksi status hunian parkir. Peneliti hanya menggunakan data gambar yang diperoleh dari penelitian sebelumnya (Amato et al. 2016). Dengan menggunakan data gambar tersebut akan dilakukan simulasi deteksi status hunian parkir. Data gambar yang telah disiapkan (kamera A sampai kamera H) akan dideteksi setiap hunian parkirnya, hasil deteksi akan dikirimkan ke database dan pada aplikasi android dapat mengambil data hasil deteksi dan menampilkan nya pada aplikasi android. Ilustrasi konsep akhir seperti pada Gambar 3.2.



Gambar 3.2. Konsep Akhir

Sehingga urutan langkah – langkah pada metodologi penelitian seperti pada Gambar 3.3, yang meliputi, pengumpulan data, pemilihan model arsitektur CNN terbaik, Rancangan sistem deteksi hunian parkir, dan pengujian.



Gambar 3.3. Langkah – langkah metodologi penelitian

Urutan langkah – langkah seperti pada flowchart di atas sebagai berikut:

1. Pengumpulan data

   Dalam pengumpulan data ini, menjelaskan bagaimana cara mendapatkan data dan bagaimana bentuk data yang digunakan, baik itu untuk pelatihan CNN atau untuk simulasi sistem deteksi hunian parkir.

2. Pemilihan model arsitektur CNN terbaik

   Pemilihan model arsitektur CNN terbaik dilakukan untuk menentukan satu model arsitektur yang memiliki nilai akurasi dan loss terbaik, yang mana model arsitektur tersebut akan digunakan untuk sistem deteksi hunian parkir pada penelitian ini.

3. Perancangan sistem deteksi hunian parkir

Perancangan sistem ini menjelaskan bagaimana sistem deteksi hunian parkir dirancang. Di dalam sistem tersebut terdapat beberapa sub-sistem seperti berikut:

a. Perancangan sistem untuk melihat hasil deteksi hunian parkir.
b. Perancangan database.
c. Perancangan sistem untuk deteksi status hunian parkir.

4. Pengujian

Pengujian ini bertujuan untuk melihat apakah sistem sudah baik atau belum, dimana beberapa pengujian dilakukan seperti pengujian untuk simulasi deteksi hunian parkir dan pengujian usability.

3.2 Pengumpulan data

Dalam penelitian ini, berdasarkan cara mendapatkan data yang digunakan dapat dikategorikan sebagai data sekunder. Data sekunder adalah data yang tidak diperoleh langsung oleh peneliti melainkan data dari hasil penelitian sebelumnya, *website*, atau perpustakaan (S. Statistik 2018). Data yang digunakan bersumber dari *cnrpark.it* dimana data tersebut memuat beberapa area parkir dengan *angle* kamera yang berbeda – beda dan data tersebut telah terlabeli (Amato et al. 2016). Pada penelitian ini, terdapat dua data yang diambil yaitu gambar yang memuat seluruh area parkir berukuran 1000x750 *pixel* (Gambar 3.4) beserta letak lokasi hunian parkirnya yang berbentuk bounding box (Tabel 3.1) dimana terdapat nilai X, Y, H, dan W. X dan Y merupakan koordinat kartesius letak hunian parkir, serta H dan W merupakan tinggi dan lebar lokasi hunian parkir tersebut. Lalu gambar per hunian parkir (Gambar 3.5) beserta label dari setiap gambar (Tabel 3.2). Data *CNRPark* sudah pernah digunakan oleh peneliti-peneliti sebelumnya, seperti (Amato et al. 2016) dan (Tanuwijaya and Fatichah 2020a).

Gambar 3.4 Contoh data gambar area parkir

Tabel 3.1. Contoh data lokasi hunian parkir

id	X (koordinat kartesius x)	Y (Koordinat kartesius y)	H (Tinggi Bounding box)	W (Lebar Bounding box)
1	2088	1570	380	373
2	1204	1560	380	380
3	264	1556	380	380
4	2050	1032	340	340
5	1723	1029	330	330

Gambar 3.5. Contoh data potongan gambar hunian parkir yang digunakan untuk training, validation, dan test

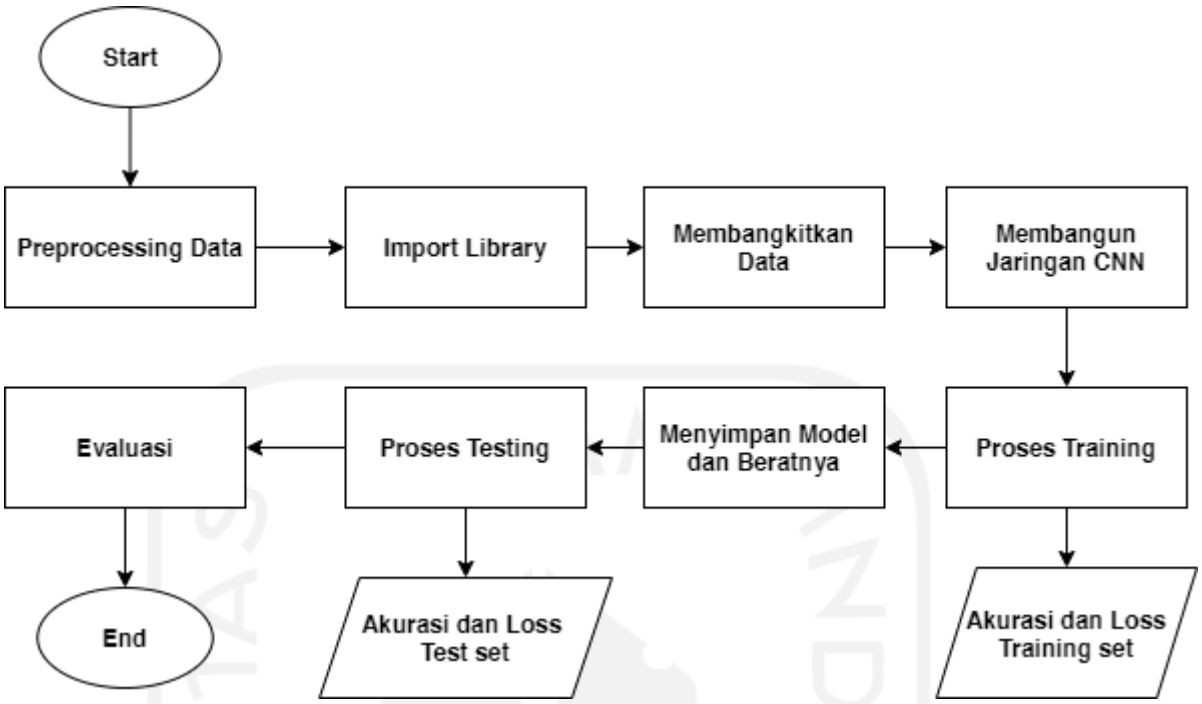
Tabel 3.2. Contoh label data

No.	file	Label	Keterangan Gambar

1	camera2/R_2016-02-12_07.35_C02_186.jpg	0	
2	camera2/R_2016-02-12_09.35_C02_187.jpg	1	
3	camera8/R_2016-01-08_08.10_C08_328.jpg	1	
4	Camera7/ S_2016-01-12_07.51_C07_210.jpg	0	

3.3 Pemilihan model arsitektur CNN terbaik

Pada Penelitian ini, sebelum membangun sistem deteksi hunian parkir terlebih dahulu akan dilakukan proses pemilihan satu model arsitektur terbaik, yang mana nantinya model arsitektur CNN ini akan digunakan untuk sistem deteksi hunian parkir. Gambar 3.6 merupakan flowchart untuk pemilihan model arsitektur terbaik.



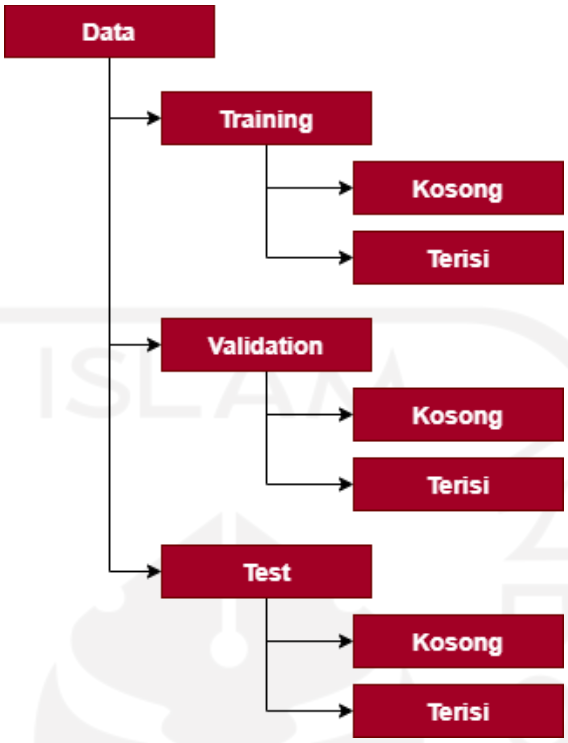
Gambar 3.6. Flowchart pemilihan model arsitektur terbaik

Pada Flowchart tersebut terdapat beberapa proses untuk mendapatkan model arsitektur terbaik, diantaranya yaitu :

1. Preprocessing data

   Sebelum dilakukan training model, terlebih dahulu dilakukan tahap *pre-processing* data. Data gambar yang didapatkan dari CNRpark sudah berlabel dengan format file csv. Data akan dibagi menjadi data *training set*, data *validation set*, dan data *test set* dimana setiap data tersebut sudah dikelompokkan sesuai status hunian seperti yang terlihat pada Gambar 3.7, pembagian data tersebut sudah dilakukan oleh penelitian sebelumnya dengan format file csv dengan kolom *path file* dan label nya. Jumlah pembagian data tersebut masing – masing yaitu, *training set* berjumlah 94492 data, *validation set* berjumlah 18646 data, dan *test set* berjumlah 31824 data.

   Training set berfungi sebagai partisi data yang digunakan untuk memperoleh bobot dari CNN, untuk menentukan bobot dari model CNN yang diperoleh dari *training set* sudah cukup, maka dilakukan proses verifikasi menggunakan *validation set.* Sedangkan *test set* berfungsi untuk memprediksi output berdasarkan model yang telah dibangun pada training dan validasi. Dengan menggunakan label data dari CNRPark serta menggunakan pyhton dapat dilakukan pengelompokan data yang berjumlah 144.965.



Gambar 3.7. Contoh Pengelompokan data

2. Import Library

Import library merupakan salah satu tahap awal untuk dapat menggunakan *functions* yang akan diperlukan pada tahap pemilihan model arsitektur CNN terbaik ini.

3. Membangkitkan data

Setelah tahap *preprocessing* data selesai, selanjutnya yaitu membangkitkan data yang telah dikelompokkan menjadi *training set*, *validation set*, dan *test set* yang nantinya akan digunakan sebagai *input* untuk proses *training* dan *testing* pada model CNN.

5. Membangun arsitektur model CNN

Pada tahap ini akan dibangun beberapa arsitektur model CNN atau menyusun lapisan – lapisan pada CNN sesuai dengan model arsitektur yang digunakan, dimana nantinya model arsitektur tersebut akan dibandingkan hasilnya. Model arsitektur tersebut adalah AlexNet, LiteAlexNet, Lenet-5, VGG16, Resnet50, GoogleNet, dan Modifikasi VGG16.

6. Proses training

   Setelah semua model arsitektur telah dibangun, selanjutnya yaitu memasuki proses *training* dimana dari proses *training* tersebut akan menghasilkan nilai akurasi dan *loss* untuk setiap model arsitektur menggunakan *training set* dan *validation set*. Model arsitektur CNN yang sudah dilakukan *training* akan disimpan dan digunakan untuk proses testing,

7. Proses testing

   Pada proses *testing* model ini, akan menggunakan *test set* sebagai *input* untuk *testing* model yang nantinya akan menghasilkan nilai akurasi dan *loss*, dimana nilai – nilai tersebut akan digunakan untuk mengevaluasi model arsitektur.

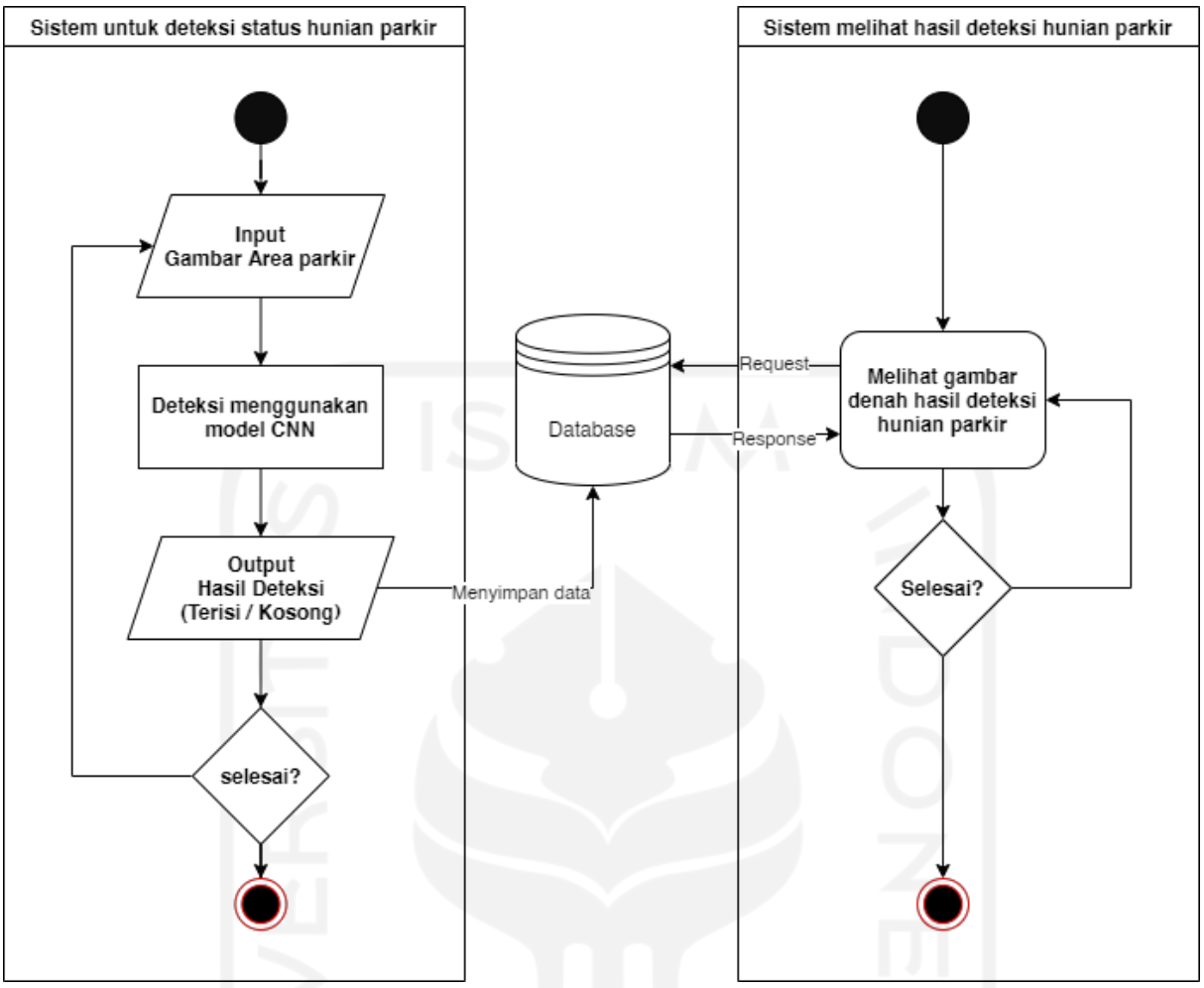
8. Evaluasi model

   Pada tahap evaluasi model CNN untuk mendeteksi status hunian parkir yaitu dengan melihat nilai akurasi dan *loss* dari hasil proses *training* dan *testing* pada setiap model arsitektur yang digunakan. Model arsitektur CNN yang memiliki nilai akurasi dan loss terbaik akan digunakan pada sistem deteksi hunian parkir.

   Dalam pengujian model CNN akan dilakukan beberapa skenario yaitu membagi *sample size* data menjadi besar, sedang, dan kecil. Pada *sample size* data besar menggunakan 100% dari jumlah data yang didapat atau sebesar 144.965 gambar, *sample size* data sedang menggunakan 70% dari jumlah data yang didapat atau sebesar 90.475 gambar, sedangkan *sample size* data kecil data menggunakan 40% dari jumlah data yang didapat atau sebesar 51.700 gambar. Serta dalam pengujian model CNN ini akan menggunakan tiga *optimizer* yang berbeda, yaitu Adam, SGD, dan RMSprop.

3.4 Perancangan sistem deteksi hunian parkir

Pada penelitian ini, Sistem deteksi hunian parkir dibagi menjadi dua bagian sub-sistem, yaitu sub-sistem untuk deteksi status hunian parkir yang ada pada sebuah komputer dan sub-sistem untuk melihat hasil dari deteksi hunian parkir yaitu aplikasi android sederhana. Gambaran sistem deteksi hunian parkir seperti pada Gambar 3.8.



Gambar 3.8. Rancangan sistem

Seperti pada Gambar 3.8, terdapat beberapa bagian pada sistem deteksi hunian parkir yaitu :

1. Sistem untuk mendeteksi status hunian parkir.

   Sistem untuk mendeteksi status hunian parkir merupakan sub-sistem yang di dalamnya terdapat proses untuk simulasi deteksi suatu gambar area parkir, dan sistem ini ada pada sebuah komputer yang dibangun menggunakan Bahasa Pemrograman Python. Pada komputer tersebut terjadi proses deteksi status hunian parkir dengan menggunakan model CNN yang telah dilatih sebelumnya, gambar area parkir yang telah disiapkan sebagai input untuk model CNN dan akan menghasilkan status hunian parkir yaitu kosong atau terisi yang diinterpretasikan dengan 0 atau 1, hasil deteksi tersebut akan disimpan pada database. Proses deteksi hunian parkir akan terus dilakukan sampai tidak ada lagi gambar area parkir yang akan dideteksi.

2. Sistem untuk melihat hasil deteksi hunian parkir.

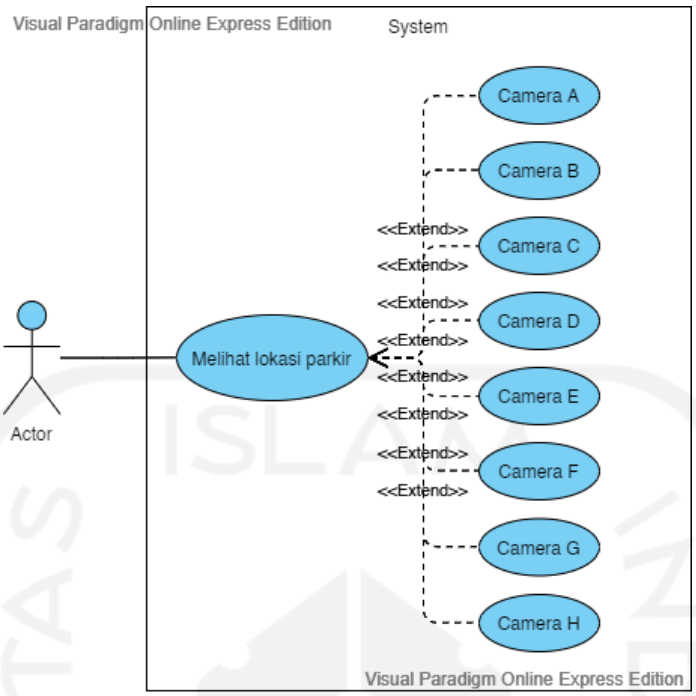
Sistem untuk melihat hasil deteksi hunian parkir merupakan suatu sub-sistem yang berfungsi hanya untuk melihat hasil dari deteksi hunian parkir. Sistem ini dibangun pada aplikasi android sederhana, yang mana *user* hanya dapat melihat denah area parkir dengan setiap hunian parkirnya memilki status hunian berupa warna, yaitu merah (terisi) dan hijau (kosong). Status hunian parkir pada aplikasi ini akan terus di-*update* selama *user* membuka aplikasi tersebut.

3.4.1 Perancangan sistem untuk melihat hasil deteksi hunian parkir

Perancangan sistem untuk melihat hasil deteksi hunian parkir pada aplikasi android ini merupakan aplikasi sederhana yang berfungsi untuk melihat hasil deteksi parkir. Dalam aplikasi ini, terdapat satu pengguna dimana dalam praktiknya pengguna hanya dapat melihat hasil deteksi hunian parkir yang divisualisasikan pada aplikasi android. Kebutuhan masukan (*input)* yaitu hasil deteksi parkir yang direpresentasikan dengan nilai 0 atau 1, sedangkan kebutuhan keluaran (output) yaitu informasi visual dari hasil deteksi parkir yang direpresentasikan dengan warna hijau atau merah.

3.4.1.1 Use case diagram

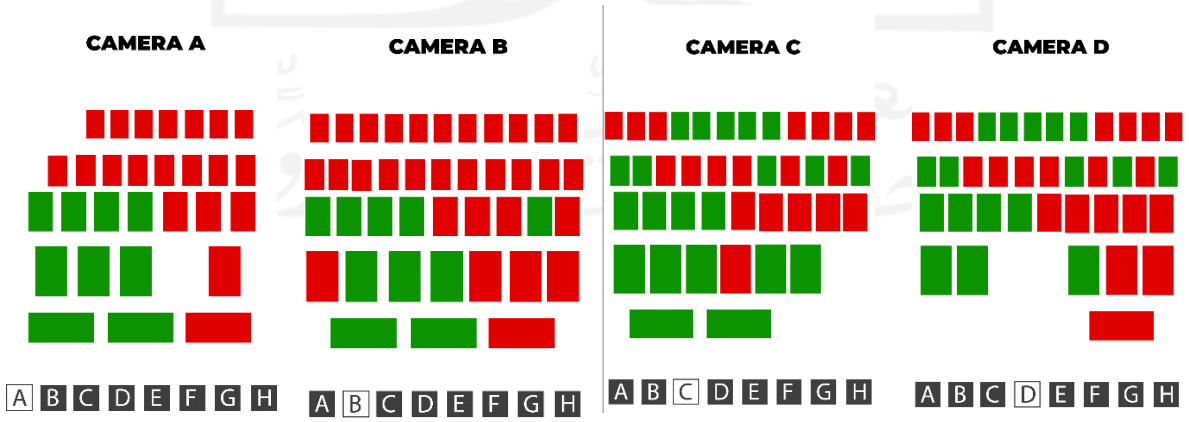
Use case diagram ini digunakan untuk melihat hubungan yang terjadi antara pengguna dengan aplikasi serta aktivitas yang dapat dilakukan, *use case diagram* dapat dilihat pada Gambar 3.9.

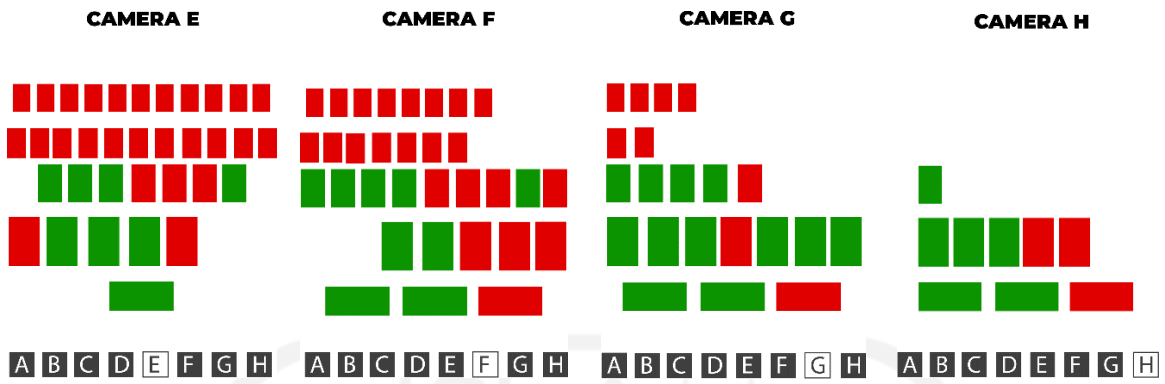


Gambar 3.9. Use Case Diagram

3.4.1.2 Perancangan antarmuka

Perancangan antarmuka pada android bertujuan memudahkan peneliti untuk membuat aplikasi android deteksi hunian parkir. Pada penelitian ini, aplikasi android terdiri dari 8 halaman yaitu CameraA, CameraB, CameraC, CameraD, CameraE, CameraF, CameraG, dan CameraH. Setiap halaman tersebut mewakili posisi camera pada gambar area parkir. Perancangan antarmuka pada aplikasi android seperti pada Gambar 3.10.

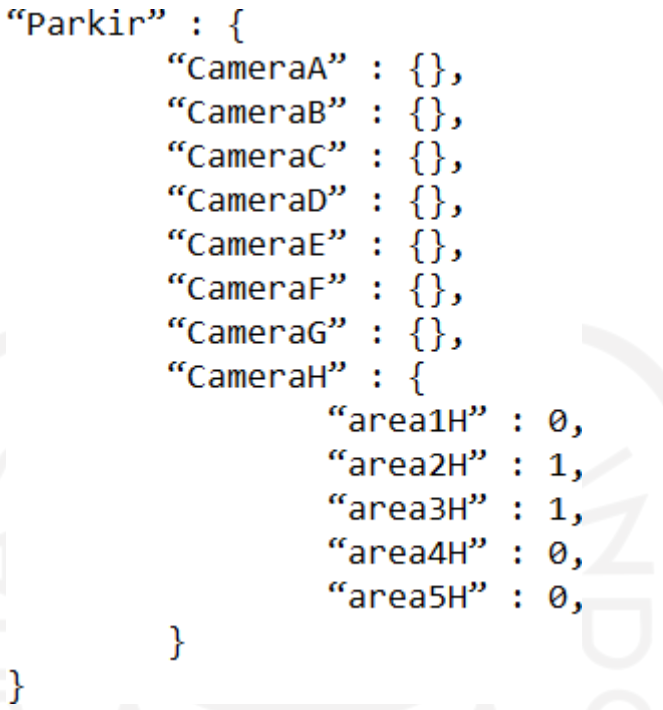




Gambar 3.10. Perancangan antarmuka

3.4.2 Perancangan database

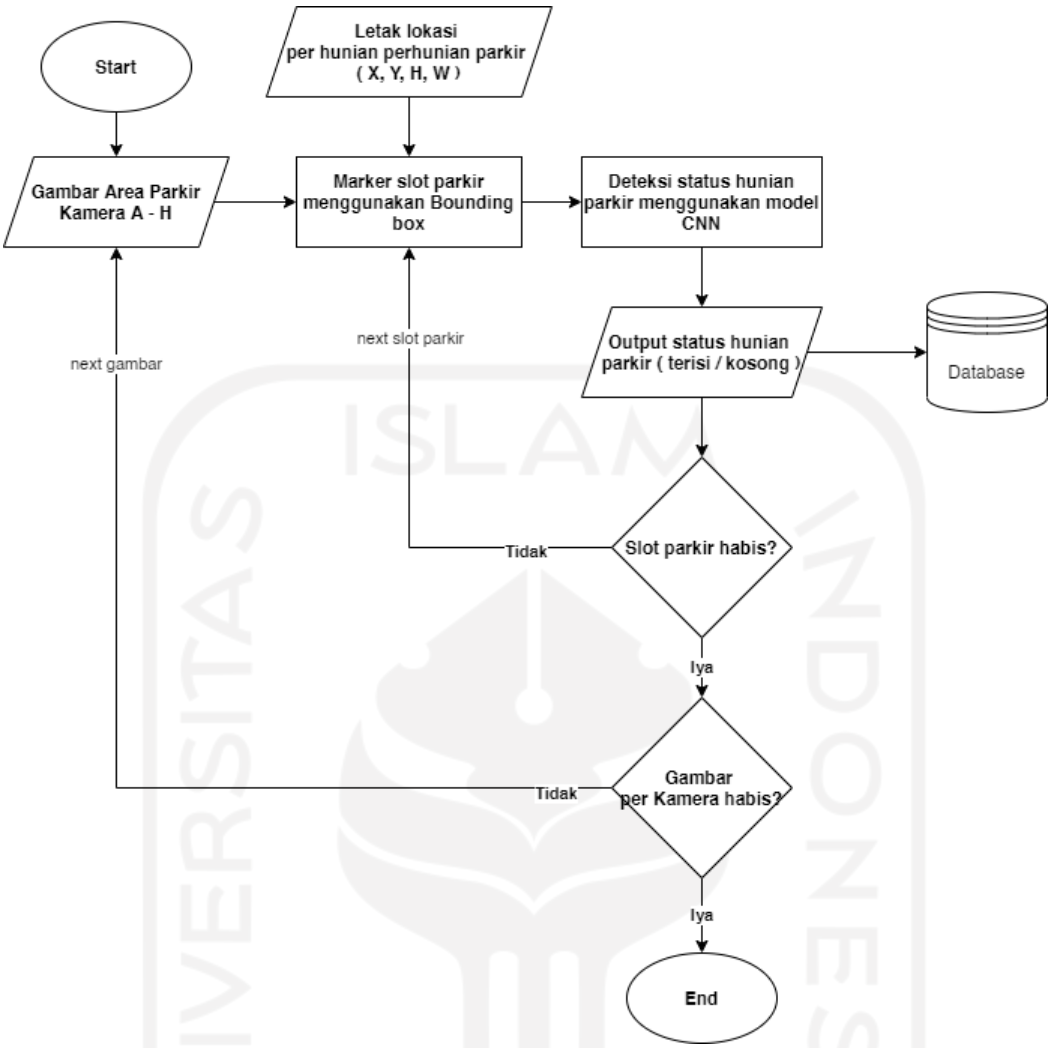
Perancangan database pada aplikasi sistem deteksi hunian parkir, dibuat berdasarkan kebutuhan masukan yang diperlukan, dimana kebutuhan masukannya adalah hasil deteksi hunian parkir pada setiap kamera di lokasi parkir. Untuk penyimpanan hasil deteksi ini akan menggunakan platform Firebase, dimana pada Firebase, struktur penyimpanan datanya menggunakan format JavaScript Object Notation (JSON), dimana JSON merupakan sebuah format data yang digunakan untuk pertukaran dan penyimpanan data. Berikut merupakan Rancangan database menggunakan format JSON seperti pada Gambar 3.11.



Gambar 3.11. Rancangan penyimpanan hasil deteksi

3.4.3 Perancangan sistem untuk deteksi status hunian parkir

Perancangan sistem untuk deteksi status hunian parkir pada sebuah komputer dibangun menggunakan Bahasa Pemrograman Python yang mana hasil deteksi tersebut akan disimpan pada database dan divisualisasikan pada aplikasi android. Tahapan untuk sistem deteksi hunian parkir dapat dilihat pada Gambar 3.12.



Gambar 3.12. Flowchart sistem untuk deteksi status hunian parkir

Flowchart pada Gambar 3.12, merupakan simulasi untuk mendeteksi gambar area parkir yang telah disiapkan (Gambar area parkir kamera A sampai kamera H), yang mana hasil dari deteksi tersebut akan langsung disimpan pada database. Dalam flowchart tersebut terdapat dua kali logika perulangan, yang pertama merupakan perulangan untuk setiap slot atau hunian parkir yang ada pada sebuah gambar area parkir, dan yang kedua merupakan perulangan untuk setiap gambar area parkir dari kamera A sampai kamera H. Proses tersebut akan terus diulang sampai tidak ada lagi gambar area parkir yang akan dideteksi.

3.5 Pengujian Sistem

Pengujian sistem pada penelitian ini bertujuan untuk mengetahui apakah sistem dapat bekerja dengan baik atau tidak. Terdapat dua tahap pengujian yaitu pengujian simulasi sistem deteksi hunian parkir dan pengujian usability.

3.5.1 Pengujian simulasi sistem deteksi hunian parkir

Pada tahap pengujian simulasi sistem deteksi hunian parkir akan menggunakan metode *black box testing.* Pengujian ini juga merupakan pengujian yang berfokus pada persyaratan fungsionalitas sistem. Pengujian ini dilakukan dengan mencoba semua kemungkinan yang terjadi dan dilakukan berulang – ulang. Jika dalam pengujian ditemukan kesalahan, maka akan dilakukan penelusuran dan perbaikan untuk memperbaiki kesalahan yang terjadi.

Skenario pengujian ini yaitu dengan menggunakan gambar area parkir dari kamera A sampai kamera H sebagai masukan untuk dideteksi, hasil dari deteksi tersebut akan dibandingkan dengan denah area parkir pada aplikasi android. Skenario pengujian sistem menggunakan metode *black box testing* dapat dilihat pada Tabel 3.3.

Tabel 3.3. Skenario pengujian simulasi deteksi hunian parkir

No.	Lokasi area parkir	Gambar area parkir	Jenis pengujian	Tipe pengujian
1	Kamera A		*Black box*	*Performance Testing*
2	Kamera B		*Black box*	*Performance Testing*
3	Kamera C		*Black box*	*Performance Testing*

4	Kamera D		*Black box*	*Performance Testing*
5	Kamera E		*Black box*	*Performance Testing*
6	Kamera F		*Black box*	*Performance Testing*
7	Kamera G		*Black box*	*Performance Testing*
8	Kamera H		*Black box*	*Performance Testing*

3.5.2 Pengujian Usability

Pengujian usability pada penelitian ini adalah dengan melakukan pemberian kuesioner kepada calon pengguna, yang mana kriteria calon pengguna adalah pengendara mobil. Pengujian ini bertujuan untuk mengetahui dan menilai apakah sistem deteksi parkir pada

perangkat android sudah sesuai dengan kebutuhan atau belum. Pengujian usability pada penelitian ini mencakup tiga aspek (Ola, Suyoto, and Purnomo 2016) yaitu :

a. *Learnability*

   Learnability yaitu kemudahan pengguna untuk memahami bagaimana menggunakan antarmuka aplikasi.

b. *Effectiveness*

   Effectiveness yaitu tingkat dukungan yang disediakan sebuah aplikasi bagi para pengguna untuk mencapai tujuannya dengan sukses.

c. *Attitude*

   Attitude merupakan tingkat kepuasan pengguna dalam menggunakan aplikasi.

Hasil pemberian kuesioner ini akan digunakan sebagai saran dalam penelitian ini. Pertanyaan dari kuesioner seperti pada Tabel 3.4, dimana masing-masing aspek terdiri dari dua pertanyaan dan terdapat satu pertanyaan dengan jawaban uraian singkat yang bersifat *optional*.

Tabel 3.4. Skenario Pengujian Usability

No	Pertanyaan	Aspek
1	Apakah aplikasi deteksi hunian parkir dirasa mudah untuk dipelajari dan digunakan	*Learnability*
2	Apakah aplikasi deteksi parkir sudah baik untuk menggambarkan kondisi dari sebuah area parkir?	
3	Apakah dengan adanya aplikasi deteksi parkir dapat memberikan bantuan informasi mengenai status hunian pada suatu area parkir?	*Effectiveness*
4	Menurut anda apakah aplikasi deteksi hunian parkir dapat mempersingkat waktu dalam mencari tempat parkir yang kosong?	
5	Apakah warna dan icon pada aplikasi sudah sesuai?	*Attitude*
6	Jika di masa yang akan datang aplikasi deteksi parkir ini sudah bisa digunakan, apakah anda akan sering menggunakannya?	
7	Saran untuk pengembangan aplikasi ini ? (optional)	

BAB IV
HASIL DAN PEMBAHASAN

41 Pemilihan model arsitektur CNN terbaik

Tahapan pemilihan model arsitektur CNN terbaik seperti pada flowchart Gambar 3.6. dimana ada beberapa tahapan yang harus dilakukan, yaitu *pre-processing data, import library,* membangkitkan data, membangun jaringan model arsitektur CNN, *training* model, *testing* model, dan evaluasi model. Pemilihan model arsitektur CNN terbaik diimplementasikan menggunakan Bahasa pemrograman Pyhton. Berikut langkah - langkah dan kode program untuk setiap proses untuk sistem deteksi hunian parkir.

4.1.1 Pre-processing data

pre-processing data pada penelitian ini yaitu membagi data menjadi *training set, validation set,* dan *test set* seperti pada Gambar 3.7 dengan menggunakan bahasa pemrograman python seperti pada Gambar 4.1.

```
import pandas as pd
import cv2

val = pd.read_csv("val.csv",sep=' ',header=None)
train = pd.read_csv("train.csv",sep=' ',header=None)
test = pd.read_csv("test.csv",sep=' ',header=None)
ntrain =94492
nval = 18646
ntest = 31824
print(train.head())

#data train
for i in range(ntrain):
    patch_train = train.loc[i,0]
    img = cv2.imread("PATCHES/" +patch_train)
    str1 = str(int(i))
    if train.loc[i,1]==1:
        cv2.imwrite('data/train/busy/train_busy'+str1+'.jpg',img)
    else:
        cv2.imwrite('data/train/free/train_free'+str1+'.jpg', img)

#data validation
for i in range(nval):
    patch_validation = val.loc[i,0]
    img = cv2.imread("PATCHES/" +patch_validation)
    str1 = str(int(i))
    if val.loc[i,1]==1:
        cv2.imwrite('data/val/busy/val_busy'+str1+'.jpg',img)
    else:
        cv2.imwrite('data/val/free/val_free'+str1+'.jpg', img)
```

```
#data test
for i in range(ntest):
    patch_test = test.loc[i,0]
    img = cv2.imread("PATCHES/" +patch_test)
    str1 = str(int(i))
    if test.loc[i,1]==1:
        cv2.imwrite('data/test/busy/test_busy'+str1+'.jpg',img)
    else:
        cv2.imwrite('data/test/free/test_free'+str1+'.jpg', img)
```

Gambar 4.1. Membagi data menjadi training set, validation set, dan test set

Terdapat dua *library* yang digunakan pada tahap pengelompokan data ini, yaitu *library* pandas untuk membaca file csv dan *library* OpenCV (cv2) untuk menyimpan file gambar sesuai dengan kelompok dan status huniannya. File csv yang berisikan label pada setiap data dibaca dan diubah menjadi data frame dengan 2 kolom yaitu kolom pertama berisikan lokasi gambar dan kolom kedua berupa status hunian (0 atau 1). Dengan menggunakan label tersebut dibuat perulangan sebanyak jumlah baris pada label, dan mengelompokkan sesuai dengan status huniannya, jika 1 maka disimpan di folder *busy* dan jika 0 maka disimpan di folder *free*.

Jumlah pembagian dataset pada penelitian ini yaitu pada *train set* 65 %, pada *validation set* 13%, dan pada *test set* 22 %. Pembagian dataset ini sudah dikelompokkan oleh sumbernya yaitu pada penelitian (Amato et al. 2016), Pembagian dataset seperti pada Tabel 4.1.

Tabel 4.1. Pembagian dataset

Distribusi data	Jumlah data
Train set	94492
Validation set	18646
Test set	31824

4.1.2 Import Library

Untuk menggunakan modul di dalam *library* pada pyhton, terlebih dahulu *library* harus diimpor diawal program seperti pada Gambar 4.2 untuk membangun model pada CNN menggunakan model Keras dari TensorFlow.

```
import keras,os
import pandas as pd
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense, Dropout,
BatchNormalization, Activation
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from tensorflow.keras.layers import Concatenate
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras import losses
from datetime import datetime
```

Gambar 4.2. *Import library*

4.1.3 Membangkitkan data

Kode pada Gambar 4.3 digunakan untuk membangkitkan sebuah *dataset*. Modul *image generator* merupakan sebuah modul untuk membangkitkan data beserta status hunian parkir. Setiap data gambar dilakukan *resize* dengan ukuran menyesuaikan kebutuhan *input* model arsitektur yang digunakan dimana pada setiap model arsitektur memiliki kebutuhan ukuran *input* yang berbeda-beda. Parameter *batch size* yang digunakan adalah 32, *shuffle true* dengan *seed* 42, serta rgb *color mode.*

```
Input_size_alexNet = (227,227
Input_size_litealexNet = (151,151)
Input_size_LeNet-5 = (28,28)
Input_size_VGG16 = (224,224)
Input_size_googleNet = (224,224)
Input_size_resNet50 = (224,224)
Input_size_modifvgg16 = (224,224)


#target_size menyesuaikan model arsitektur yang dipakai
dtrain = ImageDataGenerator()
datatrain =
dtrain.flow_from_directory(directory='./data/train',target_size=Input_size_
alexNet,batch_size=32,shuffle=True,seed=42,color_mode="rgb")

dvalidation = ImageDataGenerator()
datavalidation =
dvalidation.flow_from_directory(directory='./data/val',target_size
Input_size_alexNet,batch_size=32,shuffle=True,seed=42,color_mode="rgb")
```

Gambar 4.3. Membangkitkan data

4.1.4 Membangun jaringan CNN

Jaringan syaraf tiruan dibangun secara berurutan dengan jumlah layer sesuai dengan model arsitektur yang digunakan. Untuk memperbarui bobot secara iteratif berdasarkan data training maka kami menggunakan *adaptive moment estimation* (adam) sebagai algoritma optimisasi dengan *learning rate* 0,00001, dan identifikasi dua kelas yaitu kosong atau terisi, serta untuk optimasi nilai loss menggunakan *Crossentropy loss function*

Pada penelitian ini, peneliti menggunakan tujuh model arsitektur yang berbeda untuk dibandingkan hasilnya yaitu AlexNet seperti pada Gambar 4.4, LiteAlexNet seperti pada

Gambar 4.5, LeNet-5 seperti pada Gambar 4.6, VGG16 seperti pada Gambar 4.7, GoogleNet seperti pada Gambar 4.8, ResNet50 seperti pada Gambar 4.9, dan modifikasi VGG16 seperti pada Gambar 4.10.

```
model = Sequential()
model.add(Conv2D(filters=96, kernel_size=(11,11), strides=(4,4),
activation='relu', input_shape=(227,227,3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1),
activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=384, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(1,1), strides=(1,1),
activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(2, activation='softmax'))

opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Gambar 4.4. Membangun layer CNN model arsitektur AlexNet

Pembentukan jaringan model arsitektur AlexNet dibuat secara berurutan menggunakan modul *Sequential()* lalu dilanjutkan dengan menambahkan *layer* yang dibutuhkan dengan parameternya. Kebutuhan ukuran *input* pada AlexNet yaitu 227x227x3. *Layer's* yang digunakan pada AlexNet diantaranya yaitu pertama Conv2D (lapisan konvolusi) dengan parameter *filters* (20 dan 50), ukuran kernel, *stride, padding,* dan fungsi aktivasi, pada padding dan fungsi aktivasi yang digunakan selalu sama yaitu padding "*same*" dan aktivasi "*relu*". Kedua yaitu batch normalization sebelum dilakukannya *pooling layer*. Ketiga MaxPool2D (*pooling layer*) dengan parameter ukuran *pooling* dan *stride,* Keempat Flatten yaitu untuk mengubah lapisan-lapisan sebelumnya menjadi *fully connected layer*. Yang terakhir menambahkan 4096 *hidden layer* sebanyak dua kali dengan fungsi aktivasi yaitu "*relu"* dan diikuti klasifikasi dua kelas menggunakan fungsi aktivasi "*softmax".*

```
model = Sequential()
model.add(Conv2D(filters=20, kernel_size=(11,11), strides=(2,2),
activation='relu', input_shape=(151,151,3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3,3), strides=(2,2)))
model.add(Conv2D(filters=25, kernel_size=(5,5), strides=(3,3),
activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3,3), strides=(1,1)))
model.add(Conv2D(filters=30, kernel_size=(3,3), strides=(2,2),
activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(3,3), strides=(1,1)))

model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(30, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Gambar 4.5. Membangun layer CNN model arsitektur LiteAlexNet

Pembentukan jaringan model arsitektur LiteAlexNet dibuat secara berurutan menggunakan modul *Sequential()* lalu dilanjutkan dengan menambahkan *layer* yang dibutuhkan dengan parameternya. Kebutuhan ukuran *input* pada AlexNet yaitu 151x151x3. *Layer's* yang digunakan pada LiteAlexNet diantaranya yaitu pertama Conv2D (lapisan konvolusi) dengan parameter *filters* (20, 25, dan 30), ukuran kernel (11x11, 5x5, dan 3x3), *stride* (2x2, 3x3, dan 2x2), *padding*, dan fungsi aktivasi, pada padding dan fungsi aktivasi yang digunakan selalu sama yaitu padding "*same*" dan aktivasi "*relu*". Kedua yaitu batch normalization sebelum dilakukannya *pooling layer*. Ketiga MaxPool2D (*pooling layer*) dengan parameter ukuran *pooling* 3x3 dan *stride* 1x1, Keempat Flatten yaitu untuk mengubah lapisan-lapisan sebelumnya menjadi *fully connected layer*. Yang terakhir menambahkan 30 *hidden layer* dengan fungsi aktivasi yaitu "*relu"* dan diikuti klasifikasi dua kelas menggunakan fungsi aktivasi "*sigmoid".*

```
model = Sequential()
model.add(Conv2D(filters=6,kernel_size=(5,5),padding='valid',input_shape=(2
8,28,3),activation='tanh'))
model.add(AvgPool2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(filters=16,kernel_size=(5,5),
padding='valid',activation='tanh'))
model.add(AvgPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(120,activation='tanh'))
model.add(Dense(84,activation='tanh'))
model.add(Dense(2,activation='softmax'))

opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Gambar 4.6. Membangun layer CNN model arsitektur LeNet-5

Pembentukan jaringan model arsitektur LeNet-5 dibuat secara berurutan menggunakan modul *Sequential()* lalu dilanjutkan dengan menambahkan *layer* yang dibutuhkan dengan parameternya. Kebutuhan ukuran *input* pada LeNet-5 yaitu 28x28x3. *Layer's* yang digunakan pada LeNet-5 diantaranya yaitu pertama Conv2D (lapisan konvolusi) dengan parameter *filters* (6 dan 16), ukuran kernel, *padding*, dan fungsi aktivasi, pada padding dan fungsi aktivasi yang digunakan selalu sama yaitu padding "*valid*" dan aktivasi "*tanh*". Kedua AvgPool2D (*pooling layer*) dengan parameter ukuran *pooling* 2x2 dan *stride* (2, 2), Ketiga Flatten yaitu untuk mengubah lapisan-lapisan sebelumnya menjadi *fully connected layer*. Yang terakhir menambahkan 120 dan 84 *hidden layer* sebanyak dua kali dengan fungsi aktivasi yaitu"*tanh"* dan diikuti klasifikasi dua kelas menggunakan fungsi aktivasi "*softmax".*

```
model = Sequential()

model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),paddi
ng="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=128,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=128,kernel_size=(3,3),padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=256,kernel_size=(3,3),padding="same",
activation="relu"))
```

```
model.add(Conv2D(filters=256,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=256,kernel_size=(3,3),padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=512,kernel_size=(3,3),padding="same",
activation="relu"))

model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=2, activation="softmax"))


opt = Adam(lr=0.00001)
model.compile(optimizer=opt,loss=losses.categorical_crossentropy,
metrics=['accuracy'])

model.summary()
```

Gambar 4.7. Membangun layer CNN model arsitektur VGG16

Pembentukan jaringan model arsitektur VGG16 juga dibuat secara berurutan menggunakan modul *Sequential()* lalu dilanjutkan dengan menambahkan *layer* yang dibutuhkan dengan parameternya. Kebutuhan ukuran *input* pada VGG16 yaitu 228x228x3. Sebelum menambahkan *pooling layer*, *layer* Conv2D (lapisan konvolusi) pada VGG16 ditambahkan dua sampai 3-layer dengan parameter yang sama yaitu ukuran kernel 3x3, *padding* "*same*", dan fungsi aktivasi "*relu*", kecuali ukuran *filter* yaitu 64, 128, 256, dan 512. Setelah menambahkan dua sampai tiga *layer* Conv2D, lalu menambahkan *layer* MaxPool2D (*pooling layer)* dengan parameter ukuran *pooling* 2x2 dan *stride* 2x2. Selanjutnya yaitu Flatten untuk mengubah lapisan-lapisan sebelumnya menjadi *fully connected layer*. Dan terakhir menambahkan 4096 *hidden layer* sebanyak dua kali dengan fungsi aktivasi yaitu "*relu"* dan diikuti klasifikasi dua kelas menggunakan fungsi aktivasi "*softmax".*

```
def inception(x, filters):
    # 1x1
    path1 = Conv2D(filters=filters[0],kernel_size=(1,1),strides=1,
            padding='same', activation='relu')(x)
    # 1x1->3x3
    path2 = Conv2D(filters=filters[1][0], kernel_size=(1, 1), strides=1,
            padding='same', activation='relu')(x)
    path2 = Conv2D(filters=filters[1][1], kernel_size=(3, 3), strides=1,
            padding='same', activation='relu')(path2)
    # 1x1->5x5
    path3 = Conv2D(filters=filters[2][0], kernel_size=(1, 1), strides=1,
            padding='same', activation='relu')(x)
    path3 = Conv2D(filters=filters[2][1], kernel_size=(5, 5), strides=1,
            padding='same', activation='relu')(path3)
    # 3x3->1x1
    path4 = MaxPooling2D(pool_size=(3, 3), strides=1, padding='same')(x)
    path4 = Conv2D(filters=filters[3], kernel_size=(1, 1), strides=1,
            padding='same', activation='relu')(path4)
    return Concatenate(axis=-1)([path1, path2, path3, path4])


def auxiliary(x, name=None):
    layer = AveragePooling2D(pool_size=(5, 5), strides=3,
            padding='valid')(x)
    layer = Conv2D(filters=128, kernel_size=(1, 1), strides=1,
            padding='same', activation='relu')(layer)
    layer = Flatten()(layer)
    layer = Dense(units=256, activation='relu')(layer)
    layer = Dropout(0.4)(layer)
    layer = Dense(units=CLASS_NUM, activation='softmax', name=name)(layer)

    return layer


def googlenet():
    layer_in = Input(shape=IMAGE_SHAPE)
    # stage-1
    layer = Conv2D(filters=64, kernel_size=(7, 7), strides=2,
            padding='same', activation='relu')(layer_in)
    layer = MaxPooling2D(pool_size=(3, 3), strides=2,
            padding='same')(layer)
    layer = BatchNormalization()(layer)
    # stage-2
    layer = Conv2D(filters=64, kernel_size=(1, 1), strides=1,
            padding='same', activation='relu')(layer)
    layer = Conv2D(filters=192, kernel_size=(3, 3), strides=1,
            padding='same', activation='relu')(layer)
    layer = BatchNormalization()(layer)
    layer = MaxPooling2D(pool_size=(3, 3), strides=2,
            padding='same')(layer)
    # stage-3
    layer = inception(layer, [64, (96, 128), (16, 32), 32])  # 3a
    layer = inception(layer, [128, (128, 192), (32, 96), 64])  # 3b
    layer = MaxPooling2D(pool_size=(3, 3), strides=2,
            padding='same')(layer)
    # stage-4
    layer = inception(layer, [192, (96, 208), (16, 48), 64])  # 4a
    aux1 = auxiliary(layer, name='aux1')
```

```
    layer = inception(layer, [160, (112, 224), (24, 64), 64])  # 4b
    layer = inception(layer, [128, (128, 256), (24, 64), 64])  # 4c
    layer = inception(layer, [112, (144, 288), (32, 64), 64])  # 4d
    aux2 = auxiliary(layer, name='aux2')
    layer = inception(layer, [256, (160, 320), (32, 128), 128])  # 4e
    layer = MaxPooling2D(pool_size=(3, 3), strides=2,
            padding='same')(layer)
    # stage-5
    layer = inception(layer, [256, (160, 320), (32, 128), 128])  # 5a
    layer = inception(layer, [384, (192, 384), (48, 128), 128])  # 5b
    layer = AveragePooling2D(pool_size=(7, 7), strides=1,
            padding='valid')(layer)
    # stage-6
    layer = Flatten()(layer)
    layer = Dropout(0.4)(layer)
    layer = Dense(units=256, activation='linear')(layer)
    main = Dense(units=CLASS_NUM, activation='softmax', name='main')(layer)

    model = Model(inputs=layer_in, outputs=[main, aux1, aux2])

    return model

model = googlenet()
opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Gambar 4.8. Membangun layer CNN model arsitektur GoogleNet

Pada arsitektur GoogleNet terdapat 3 fungsi yaitu inception() dengan parameter x sebagai *input* dan *filter* sebagai array yang menyimpan ukuran *filter*, auxiliary() dengan parameter x sebagai *input*, dan fungsi googlenet() untuk membangun jaringan CNN.

Pada fungsi inception() terdiri dari perkalian antara input berupa *feature map* hasil lapisan sebelumnya dan *path* berupa Conv2D dan MaxPool2D, hasil perkalian tersebut pada akhirnya ditambah dan menghasilkan *inception module.*

Fungsi auxiliary() terdiri dari AveragePooling2D (dengan parameter ukuran pooling 2x2, *padding* "*valid*", dan *stride* 3), Conv2D (dengan parameter *filter* 128, ukuran kernel 1x1, *stride* 1, *padding* "*same*" dan fungsi aktivasi "relu"), Flatten() untuk mengubah lapisan sebelumnya menjadi *fully connected layer*, menambahkan *hidden layer* sebanyak 256 dengan fungsi aktivasi "*relu*", dan fungsi aktivasi "*softmax"* untuk klasifikasi.

Pada fungsi googlenet() terdiri dari enam *stages* yang tersusun secara berurutan, sebelum memasuki enam *stages* tersebut ditambahkan *layer* untuk *input* gambar (224x224x3). *Stage* pertama dan kedua terdiri dari Conv2D, MaxPool2D, dan BatchNormalization. *Stage* ketiga terdiri dari fungsi inception() dan MaxPooling2D. *Stage* keempat terdiri dari fungsi inception(), auxiliary(), dan MaxPooling2D. *Stage* kelima terdiri dari fungsi inception() dan

AveragePooling2D. Dan *stage* terakhir terdiri dari Flatten. Dropout layer dengan parameter 0,4, Penambahan hidden layer sebanyak 256-unit dengan aktivasi "*linear*", dan klasifikasi dengan fungsi aktivasi "*softmax*".

```
def res_identity(x, filters):
    x_skip = x # this will be used for addition with the residual block
    f1, f2 = filters

    #first block
    x = Conv2D(f1, kernel_size=(1, 1), strides=(1, 1), padding='valid',
kernel_initializer='he_normal')(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    #second block # bottleneck (but size kept same with padding)
    x = Conv2D(f1, kernel_size=(3, 3), strides=(1, 1), padding='same',
kernel_initializer='he_normal')(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    # third block activation used after adding the input
    x = Conv2D(f2, kernel_size=(1, 1), strides=(1, 1), padding='valid',
kernel_initializer='he_normal')(x)
    x = BatchNormalization()(x)
    # x = Activation(activations.relu)(x)
    # add the input
    x = Add()([x, x_skip])
    x = Activation("relu")(x)

    return x

def res_conv(x, s, filters):
    x_skip = x
    f1, f2 = filters

    # first block
    x = Conv2D(f1, kernel_size=(1, 1), strides=(s, s), padding='valid',
kernel_initializer='he_normal')(x)
    # when s = 2 then it is like downsizing the feature map
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    # second block
    x = Conv2D(f1, kernel_size=(3, 3), strides=(1, 1), padding='same',
kernel_initializer='he_normal')(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    #third block
    x = Conv2D(f2, kernel_size=(1, 1), strides=(1, 1), padding='valid',
kernel_initializer='he_normal')(x)
    x = BatchNormalization()(x)
    # shortcut
    x_skip = Conv2D(f2, kernel_size=(1, 1), strides=(s, s),
padding='valid', kernel_initializer='he_normal')(x_skip)
    x_skip = BatchNormalization()(x_skip)
    # add
    x = Add()([x, x_skip])
    x = Activation("relu")(x)

    return x
```

```
def resnet50():
    input_im = Input_size_resNet50
    x = ZeroPadding2D(padding=(3, 3))(input_im)

    # 1st stage
    x = Conv2D(64, kernel_size=(7, 7), strides=(2, 2))(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPooling2D((3, 3), strides=(2, 2))(x)
    #2nd stage
    x = res_conv(x, s=1, filters=(64, 256))
    x = res_identity(x, filters=(64, 256))
    x = res_identity(x, filters=(64, 256))
    # 3rd stage
    x = res_conv(x, s=2, filters=(128, 512))
    x = res_identity(x, filters=(128, 512))
    x = res_identity(x, filters=(128, 512))
    x = res_identity(x, filters=(128, 512))
    # 4th stage
    x = res_conv(x, s=2, filters=(256, 1024))
    x = res_identity(x, filters=(256, 1024))
    x = res_identity(x, filters=(256, 1024))
    x = res_identity(x, filters=(256, 1024))
    x = res_identity(x, filters=(256, 1024))
    x = res_identity(x, filters=(256, 1024))
    # 5th stage
    x = res_conv(x, s=2, filters=(512, 2048))
    x = res_identity(x, filters=(512, 2048))
    x = res_identity(x, filters=(512, 2048))

    x = AveragePooling2D((2, 2), padding='same')(x)
    x = Flatten()(x)
    x = Dense(2, activation='softmax', kernel_initializer='he_normal')(x)
#multi-class

    model = Model(inputs=Input_size_resNet50, outputs=x, name='Resnet50')

    return model

model = resnet50()
opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Gambar 4.9. Membangun layer CNN model arsitektur ResNet50

Pada arsitektur ResNet50 terdapat 3 fungsi yaitu res_identity() dengan parameter x sebagai *input* dan *filter* sebagai array yang menyimpan ukuran *filter*, res_conv() dengan parameter x sebagai *input*, s sebagai ukuran *stride* dan fungsi resnet50() untuk membangun jaringan CNN. Fungsi res_identity() dan res_conv() merupakan fungsi untuk menjalankan konsep *skip connection*, yang membedakan adalah pada res_identity() ukuran *stride* selalu sama yaitu 1x1 sedangkan pada res_conv() ukuran *stride* tidak selalu sama.

Pada fungsi resnet50() terdiri dari lima *stages*, kebutuhan *input* pada ResNet50 adalah 224x224x3 dan ditambahkan *zero padding* dengan ukuran 3x3. Pada *stage* pertama terdiri dari *layer* Conv2D , BatchNormalization, Activation, dan MaxPooling2D. Sedangkan pada *stage* dua sampai lima terdiri dari fungsi res_identity() dab res_conv(), dan yang terakhir menambahkan AveragePolling2D dan Flatten untuk dapat dilakukan klasifikasi 2 kelas dengan menggunakan aktivasi "*softmax*".

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=32,kernel_size=(3,3),paddi
ng="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=64, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=2, activation="softmax"))

opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss=losses.categorical_crossentropy,
metrics=['accuracy'])
model.summary()
```

Gambar 4.10. Membangun layer CNN model arsitektur modifikasi VGG16

Model arsitektur modifikasi VGG16 tidak jauh berbeda dengan VGG16, yang membedakan hanyalah jumlah lapisan. Pembentukan jaringan model arsitektur modifikasi VGG16 juga dibuat secara berurutan menggunakan modul *Sequential()* lalu dilanjutkan dengan menambahkan *layer* yang dibutuhkan dengan parameternya. Kebutuhan ukuran *input* pada modifikasi VGG16 yaitu 224x224x3. Lalu dilanjutkan *layer* Conv2D (lapisan konvolusi) dengan parameter yang sama yaitu ukuran kernel 3x3, *padding* "*same*", dan fungsi aktivasi "*relu*", kecuali ukuran *filter* yaitu 32, 64, 128, 256, dan 512, Setiap lapisan konvolusi selalu diikuti dengan *layer* MaxPool2D (*pooling layer)* dengan parameter ukuran *pooling* 2x2 dan *stride* 2x2. Selanjutnya yaitu Flatten untuk mengubah lapisan-lapisan sebelumnya menjadi

fully connected layer. Dan terakhir menambahkan 1000 *hidden layer* dengan fungsi aktivasi yaitu "*relu"* dan diikuti klasifikasi dua kelas menggunakan fungsi aktivasi "*softmax".*

4.1.5 Training Model

Kode pada Gambar 4.11 merupakan proses *training* model dengan menggunakan 15 *epoch*. Untuk melihat hasil akurasi dan loss model digunakan *visualize tools* yang disediakan oleh TensorFlow yang bernama Tensorboard. Hasil *training* model disimpan untuk digunakan pada proses *testing model* menggunakan *test set*.

```
#simpan model dan berat
"""
 Tensorboard log
 """
log_dir = './tf-log/'+datetime.now().strftime("%Y%m%d-%H%M%S")
tb_cb = callbacks.TensorBoard(log_dir=log_dir, histogram_freq=0)
cbks = [tb_cb]

#pelatihan model
model.fit(datatrain,
epochs=15,validation_data=datavalidation,callbacks=cbks)

#simpan model dan berat
target_dir = './models/'
if not os.path.exists(target_dir):
    os.mkdir(target_dir)
    model.save('./models/models.h5')
    model.save_weights('./models/weights.h5')
```

Gambar 4.11. Source code training model

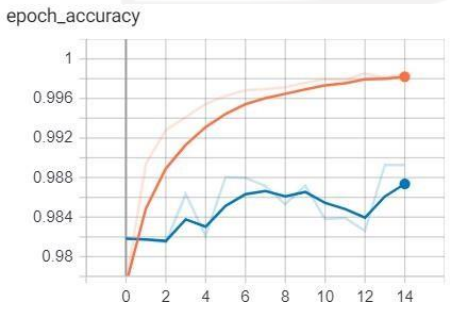
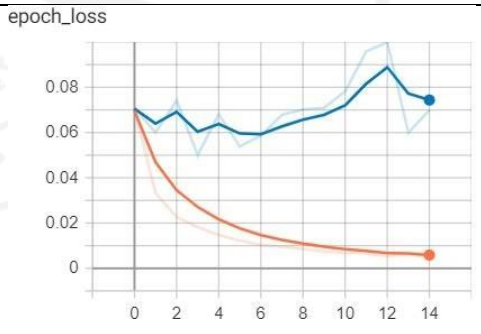
Hasil dari proses *training* pada setiap model arsitektur seperti pada Tabel 4.2. Dari hasil proses training model tersebut, menunjukkan bahwa nilai akurasi dan *loss* pada setiap model mendapatkan hasil yang sangat bagus. Cukup sulit untuk menentukan model arsitektur yang memiliki nilai akurasi dan *loss* terbaik, dikarenakan perbedaan nilai akurasi pada setiap model tersebut memiliki selisih yang sangat sedikit. Akurasi dengan nilai terendah ada pada model arsitektur LeNet-5 yaitu 94,30% pada *training set* dan 94,58% *validation set*, sedangkan akurasi tertinggi pada *training set* ada pada model arsitektur VGG16 yaitu 99,90%, pada *validation set* akurasi tertinggi ada pada modifikasi VGG16 yaitu 99,89%. Namun dari hasil yang sangat baik tersebut, masih ada kemungkinan terjadi *overfitting* jika hasil akurasi pada *testing* model terlalu kecil. Waktu yang dibutuhkan untuk melakukan proses pelatihan dengan menggunakan 15 *epoch* yaitu kurang lebih empat sampai lima hari dengan menggunakan server hanabi-01 v.1.2.0 dengan fitur 80 Cores, 250GB RAM, dan 4GPU.

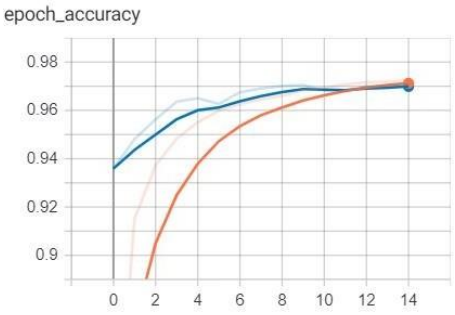
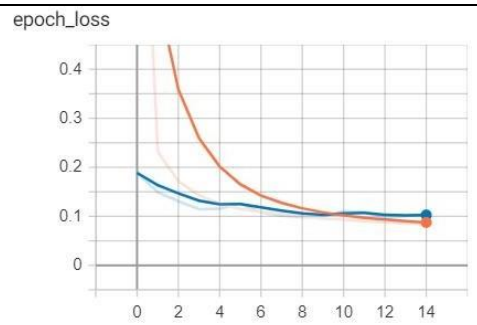
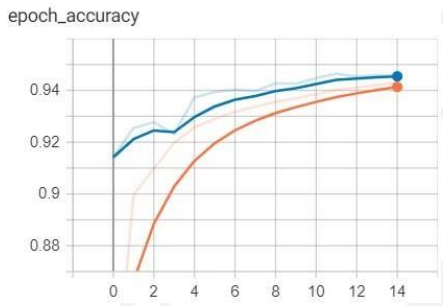
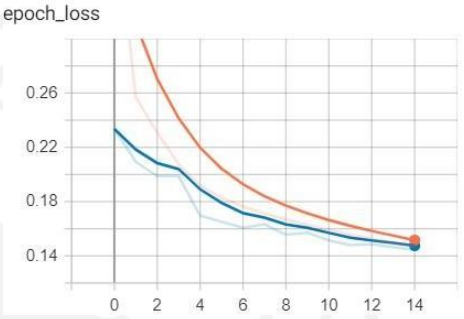
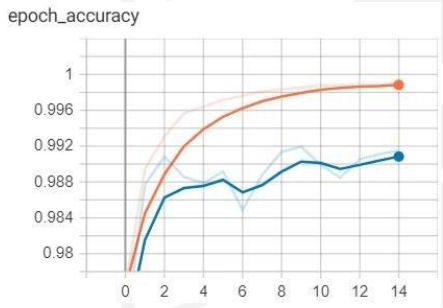
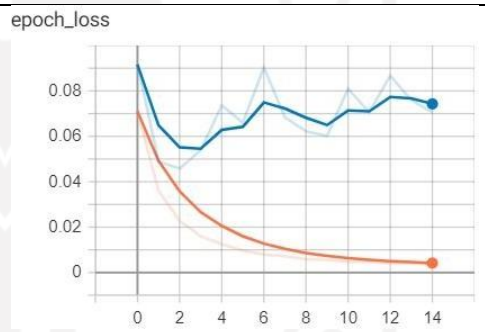
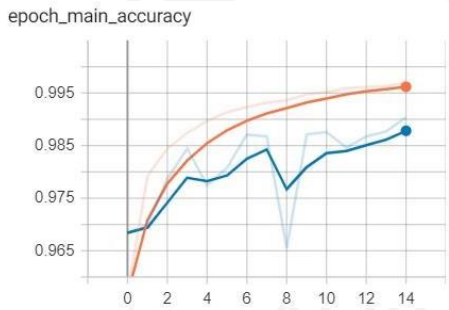
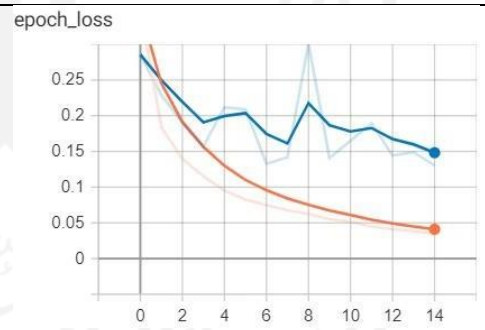
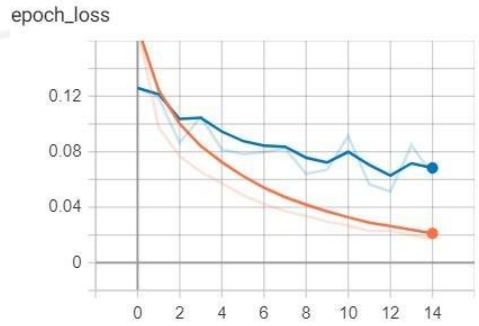
Tabel 4.2. Hasil Training model

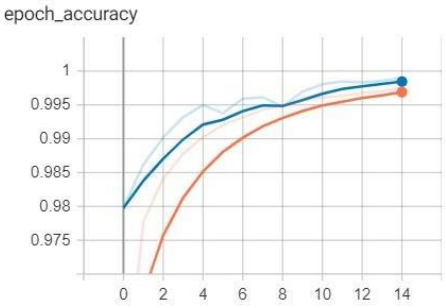
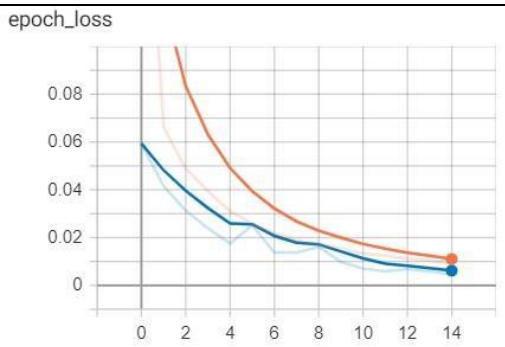
No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*
1	AlexNet	0,9984	0,0048	0,9892	0,0699
2	LiteAlexNet	0,9723	0,0837	0,9706	0,1036
3	LeNet-5	0,9430	0,1467	0,9458	0,1445
4	VGG16	**0,9990**	**0,0034**	0,9915	0,0705
5	GoogleNet	0,9968	0,0352	0,9903	0,1305
6	ResNet50	0,9937	0,0171	0,9847	0,0633
7	Modifikasi VGG16	0,9975	0,0090	**0,9989**	**0,0045**

Untuk melihat proses peningkatan atau penurunan nilai akurasi dan *loss* pada *training model*, peneliti menggunakan *visualize tools* yang disediakan oleh TensorFlow yaitu Tensorboard. Hasil dari proses training model dapat dilihat pada grafik Tensorboard seperti pada Tabel 4.3. Terdapat empat kolom yang menunjukan nama model arsitektur, grafik akurasi, grafik *loss*, dan keterangan warna pada setiap data. Grafik akurasi pada setiap model arsitektur selalu mengalami peningkatan, sedangkan pada grafik *loss* selalu mengalami penurunan.

Tabel 4.3. Grafik proses training model CNN

Model Arsitektur	Grafik Akurasi	Grafik Loss	keterangan
AlexNet			Train set Validation set

LiteAlexNet			
LeNet-5			
VGG16			
GoogleNet			
ResNet50			

Modifikasi VGG16			

4.1.6 Testing Model

Kode pada Gambar 4.12 merupakan proses *testing* model dengan menggunakan *test set* yang telah disiapkan sebelumnya. model hasil pelatihan digunakan kembali untuk proses *testing*, dimana model dan beratnya dipanggil menggunakan modul *load model* dan *load weights*. Selama proses *testing*, peneliti menggunakan modul *time* untuk menghitung berapa waktu yang dibutuhkan untuk setiap model arsitektur dalam melakukan *testing* dengan menggunakan modul evaluate().

```
model_path = 'models.h5’
model = load_model(model_path)
model.load_weights(‘weight.h5’)

dtest = ImageDataGenerator()
dataTest =
dtest.flow_from_directory(directory='test',target_size=Input_size,batch_siz
e=32,shuffle=True,seed=42,color_mode="rgb")

t = time.process_time()
test_loss, test_acc = model.evaluate(dataTest, verbose=2)
duration =time.process_time()-t

print('\nTest accuracy:', test_acc,'\nTest loss:', test_loss)
print("durasi : "+str(duration))
```

Gambar 4.12. Source code testing model

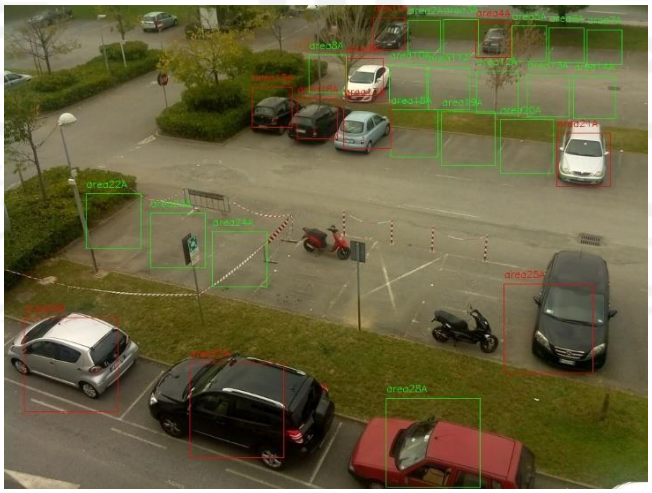
Untuk hasil proses *testing* pada setiap model arsitektur seperti pada Tabel 4.4. Dari hasil proses *testing* model tersebut, menunjukkan bahwa hasil *testing* model memperoleh nilai akurasi dan *loss* yang sangat baik dan tidak terjadi *overfitting.* Akurasi terendah ada pada model arsitektur LeNet-5 yaitu 91,92% dan juga pada model arsitektur ini memiliki waktu tercepat yaitu 67 detik. Untuk akurasi dan *loss* tertinggi ada pada model arsitektur GoogleNet yaitu 99,22% dan 0,0326 dengan pesaing terdekatnya AlexNet, VGG16, dan modifikasi VGG16 dimana hanya memiliki selisih nilai akurasi yang sangat sedikit.

Tabel 4.4. Hasil testing model

No.	Model Arsitektur	Akurasi pada *test set*	Loss pada *test set*	Waktu *test set* (Detik)
1.	AlexNet	0,9913	0,0459	4430
2.	LiteAlexNet	0,9781	0,0647	424
3.	LeNet-5	0,9192	0,1273	**67**
4.	VGG16	0,9920	0,0414	79889
5.	GoogleNet	**0,9922**	**0,0326**	9449
6.	ResNet50	0,9855	0,0510	22991
7.	Modifikasi VGG16	0,9909	0,0365	3969

Kolom waktu pada tabel diatas merupakan keterangan waktu ketika melakukan perhitungan akurasi dan loss pada *test set*, dengan satuan detik.

Perhitungan jumlah hunian parkir pada yang tepat dideteksi atau tidak pada gambar area parkir dapat menggunakan *confusion matrix* dengan mencari nilai *True Positif* (TP), *True Negative* (TN), *False Positif* (FP), *False Negative* (FN). Contoh perhitungan jumlah hasil deteksi hunian parkir menggunakan gambar area parkir (Gambar 4.13) seperti pada Tabel 4.5.



Gambar 4.13. Hasil deteksi pada gambar area parkir

Tabel 4.5. Contoh hasil confusion matrix

Predicted	Actual

		1	0
	1	17	1
	0	0	10

Setelah mendapatkan nilai TP, TN, FP, dan FN. Nilai akurasi dapat dihitung seperti pada persamaan (4.1).

$$Akurasi = \frac{17 + 10}{17 + 10 + 1 + 0} = 0{,}9643 \quad (4.1)$$

4.1.7 Evaluasi Model

Setelah melakukan proses *training* dan *testing* model didapatkan hasil akurasi yang terbilang sangat baik. Untuk menentukan model arsitektur mana yang memiliki nilai akurasi dan *loss* terbaik cukuplah sulit, dikarenakan terdapat kelebihan dan kekurangannya masing-masing pada setiap model arsitektur serta perbedaan nilai akurasi dan *loss* dengan selisih yang sangat sedikit.

Beberapa skenario dilakukan seperti pembagian sampel data yaitu *sample size* data besar menggunakan 100% dari jumlah data yang didapat atau sebesar 144.965 gambar, *sample size* data sedang menggunakan 70% dari jumlah data yang didapat atau sebesar 90.475 gambar, sedangkan *sample size* data kecil data menggunakan 40% dari jumlah data yang didapat atau sebesar 51.700 gambar. Pada pembagian *sample size* data ini, tidak ada perubahan pada parameter model, tetapi hanya mengubah jumlah datanya saja.

Serta dalam pengujian model CNN ini menggunakan tiga *optimizer* yang berbeda, yaitu Adam, SGD, dan RMSprop. Dalam penggunaan algoritma *optimizer* hanya merubah satu parameter *optimizer* sedangkan untuk *learning rate* (lr) tetap sama yaitu 0,00001 seperti pada Gambar 4.14.

```
# algoritma optimizer Adam
opt = Adam(lr=0.00001)
model.compile(optimizer=opt,loss=losses.categorical_crossentropy,metrics=['
accuracy'])

# algoritma optimizer SGD
opt = SGD(lr=0.00001)
model.compile(optimizer=opt,loss=losses.categorical_crossentropy,metrics=['
accuracy'])

# algoritma optimizer RMSprop
```

```
opt = RMSprop(lr=0.00001)
model.compile(optimizer=opt,loss=losses.categorical_crossentropy,metrics=['
accuracy'])
```

Gambar 4.14. Source code perubahan parameter pada algoritma optimizer

merupakan hasil dari skenario pembagian sampel data dan algoritma optimasi yang berbeda-beda.

4.1.7.1 Sample size data besar

Pada *sample size* data besar ini, model arsitektur yang memiliki nilai akurasi dan *loss* diatas 99% pada semua data set adalah VGG16, GoogleNet, dan Modifikasi VGG16, namun dari ketiga arsitektur tersebut Modifikasi VGG16 memiliki durasi waktu tercepat dan memiliki akurasi pada *validation set* terbaik yaitu 99,89% serta memiliki nilai *loss* terendah pada *validation set* yaitu 0,0045. Sedangkan pada training set nilai akurasi dan loss terbaik ada pada VGG16 dengan nilai 99,9% dan 0,0034. Pada test set nilai akurasi dan loss terbaik ada pada GoogleNet dengan nilai 99,22% dan 0,0326.

Tabel 4.6. Hasil training dan testing model CNN sample size data besar

No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*	Akurasi *test set*	Loss *test set*	Waktu (Detik)
1	AlexNet	0,9984	0,0048	0,9892	0,0699	0,9913	0,0459	4430
2	LiteAlexNet	0,9723	0,0837	0,9706	0,1036	0,9781	0,0647	424
3	LeNet-5	0,9430	0,1467	0,9458	0,1445	0,9192	0,1273	67
4	VGG16	**0,9990**	**0,0034**	0,9915	0,0705	0,9920	0,0414	79889
5	GoogleNet	0,9968	0,0352	0,9903	0,1305	**0,9922**	**0,0326**	9449
6	ResNet50	0,9937	0,0171	0,9847	0,0633	0,9855	0,0510	22991
7	Modifikasi VGG16	0,9975	0,0090	**0,9989**	**0,0045**	0,9909	0,0365	3969

4.1.7.2 Sample size data sedang

Pada *sample size* data sedang ini, model arsitektur yang memiliki nilai akurasi dan *loss* terbaik pada *training set* dan *test set* yaitu VGG16 dengan nilai pada *training set* 99,91% dan 0,0034 serta pada *test set* 99,70% dan 0,0161, pada *validation set* model arsitektur yang memiliki nilai akurasi dan *loss* terbaik yaitu Modifikasi VGG16 dengan

nilai 99,85% dan 0,0050. Dari hasil tersebut, terdapat dua model arsitektur terbaik yaitu VGG16 dan Modifikasi VGG16 tetapi jika dilihat dari durasi proses *testing* model, Modifikasi VGG16 memiliki durasi waktu lebih cepat dibandingkan dengan VGG16.

Tabel 4.7 Hasil training dan testing model CNN sample size data sedang

No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*	Akurasi *test set*	Loss *test set*	Waktu (Detik)
1	AlexNet	0,9984	0,0050	0,9923	0,0310	0,9931	0,0295	995
2	LiteAlexNet	0,9764	0,0708	0,9779	0,0667	0,9777	0,0696	89
3	LeNet-5	0,9932	0,0219	0,9763	0,0810	0,9743	0,0768	**33**
4	VGG16	**0,9991**	**0,0034**	0,9969	0,0164	**0,9970**	**0,0161**	11367
5	GoogleNet	0,9965	0,0365	0,9933	0,0745	0,9931	0,0757	2232
6	ResNet50	0,9925	0,0215	0,9766	0,0666	0,9750	0,0689	7585
7	Modifikasi VGG16	0,9975	0,0089	**0,9985**	**0,0050**	0,9958	0,0166	1286

4.1.7.3 Sample size data kecil

Pada *sample size* data kecil ini, model arsitektur yang memiliki nilai akurasi dan *loss* terbaik pada *training set* yaitu VGG16 dengan nilai 99,89% dan 0,0041, pada *validation set* model arsitektur yang memiliki nilai akurasi dan *loss* terbaik yaitu Modifikasi VGG16 dengan nilai 99,84% dan 0,0070, pada *test set* nilai akurasi terbaik ada pada VGG16 dengan nilai 99,51% sedangkan nilai *loss* terbaik ada pada Modifikasi VGG16 dengan nilai 0,0181. Dari hasil tersebut, terdapat dua model arsitektur terbaik yaitu VGG16 dan Modifikasi VGG16 tetapi jika dilihat dari durasi proses *testing* model, Modifikasi VGG16 memiliki durasi waktu lebih cepat dibandingkan dengan VGG16.

Tabel 4.8. Hasil training dan testing model CNN sample size data kecil

No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*	Akurasi *test set*	Loss *test set*	Waktu (Detik)
1	AlexNet	0,9985	0,0048	0,9929	0,0466	0,9938	0,0197	692

2	LiteAlexNet	0,9673	0,0977	0,9699	0,0974	0,9700	0,0866	73
3	LeNet-5	0,9918	0,0267	0,9839	0,0588	0,9860	0,0429	**27**
4	VGG16	**0,9989**	**0,0041**	0,9944	0,0358	**0,9951**	0,0223	7785
5	GoogleNet	0,9943	0,0557	0,9865	0,1985	0,9860	0,1860	1358
6	ResNet50	0,9896	0,0293	0,9824	0,0641	0,9800	0,0627	5159
7	Modifikasi VGG16	0,9968	0,0110	**0,9984**	**0,0070**	0,9936	**0,0181**	881

4.1.7.1 SGD

Pada hasil proses *training* dan *testing* menggunakan *optimizer SGD* dan menggunakan *sample size* data kecil, model arsitektur yang memiliki nilai akurasi dan *loss* terbaik pada *training set, validation set, test set* yaitu AlexNet dengan nilai 97,19% dan 0,0838 pada *training set*, 97,19% dan 0,0903 pada *validation set,* serta 97,23% dan 0,0815 pada *test set.*

Tabel 4.9. Hasil training dan testing model CNN sample size data kecil dengan algoritma optimasi SGD

No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*	Akurasi *test set*	Loss *test set*	Waktu (Detik)
1	AlexNet	**0,9719**	**0,0838**	**0,9719**	**0,0903**	**0,9723**	**0,0815**	563
2	LiteAlexNet	0,9343	0,1862	0,9494	0,1488	0,9508	0,1402	61
3	LeNet-5	0,9609	0,1104	0,9560	0,1330	0,9605	0,1220	19
4	VGG16	0,9647	0,0946	0,9606	0,1014	0,9632	0,0951	7306
5	GoogleNet	0,9036	0,7909	0,9218	0,7241	0,9238	0,7185	1434
6	ResNet50	0,8980	0,2541	0,8897	0,2847	0,8851	0,2807	4929
7	Modifikasi VGG16	0,9543	0,1218	0,9553	0,1171	0,9539	0,1248	772

4.1.7.2 RMSprop

Pada hasil proses *training* dan *testing* menggunakan *optimizer RMSprop* dan menggunakan *sample size* data kecil, model arsitektur yang memiliki nilai akurasi dan *loss* terbaik pada *training set* yaitu AlexNet dengan nilai 99,77% dan 0,0087, pada *validation*

set model arsitektur yang memiliki nilai akurasi dan *loss* terbaik yaitu Modifikasi VGG16 dengan nilai 99,7% dan 0,0151 pada *test set* nilai akurasi dan *loss* terbaik ada pada Modifikasi VGG16 dengan nilai 99,38% dan 0,0265. Dari hasil tersebut, model arsitektur terbaik dengan menggunakan *optimizer RMSprop* yaitu Modifikasi VGG16

Tabel 4.10.Hasil training dan testing model CNN sample size data kecil dengan algoritma optimasi RMSprop

No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*	Akurasi *test set*	Loss *test set*	Waktu (Detik)
1	AlexNet	**0,9977**	**0,0087**	0,9910	0,1172	0,9919	0,0723	624
2	LiteAlexNet	0,9665	0,0971	0,9715	0,0913	0,9739	0,0889	66
3	LeNet-5	0,9900	0,0336	0,9838	0,0581	0,9851	0,0446	18
4	VGG16	0,9960	0,0217	0,9890	0,0947	0,9905	0,0457	13658
5	GoogleNet	0,9948	0,0574	0,9911	0,1297	0,9916	0,0748	1483
6	ResNet50	0,9895	0,0315	0,9810	0,0630	0,9814	0,0566	3931
7	Modifikasi VGG16	0,9929	0,0283	**0,9970**	**0,0151**	**0,9938**	**0,0265**	923

4.1.7.3 Adam

Pada hasil proses *training* dan *testing* menggunakan *optimizer Adam* sama dengan hasil *training* dan *testing* menggunakan *sample size* data kecil, dikarenakan pada proses tersebut juga menggunakan *optimizer Adam* yang mana model arsitektur yang memiliki nilai akurasi dan *loss* terbaik pada *training set* yaitu VGG16 dengan nilai 99,89% dan 0,0041, pada *validation set* model arsitektur yang memiliki nilai akurasi dan *loss* terbaik yaitu Modifikasi VGG16 dengan nilai 99,84% dan 0,0070, pada *test set* nilai akurasi terbaik ada pada VGG16 dengan nilai 99,51% sedangkan nilai *loss* terbaik ada pada Modifikasi VGG16 dengan nilai 0,0181. Dari hasil tersebut, terdapat dua model arsitektur terbaik yaitu VGG16 dan Modifikasi VGG16 tetapi jika dilihat dari durasi proses *testing* model, Modifikasi VGG16 memiliki durasi waktu lebih cepat dibandingkan dengan VGG16.

Tabel 4.11. Hasil training dan testing model CNN sample size data kecil dengan algoritma optimasi Adam

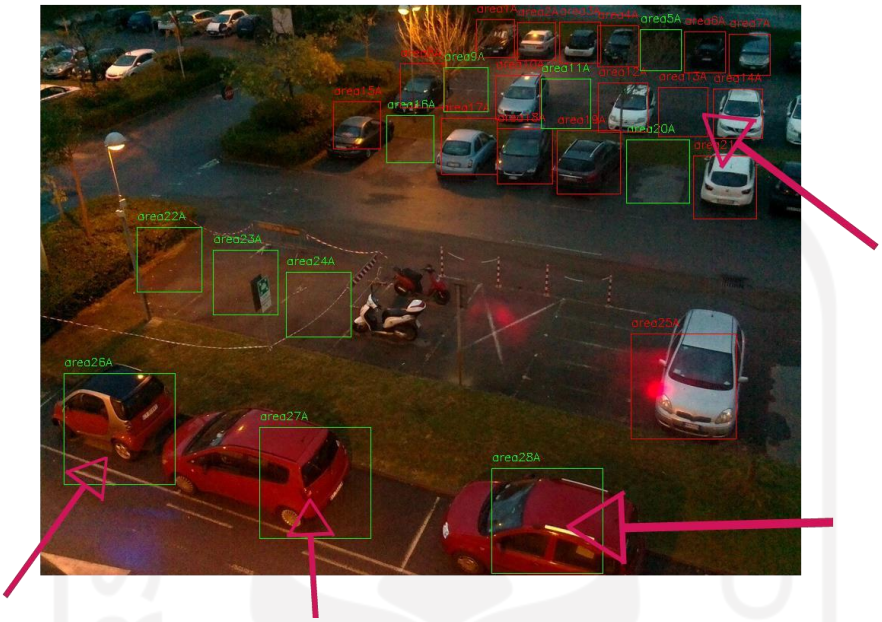
No.	Model arsitektur	Akurasi *training set*	Loss *training set*	Akurasi validation set	Loss *validation set*	Akurasi *test set*	Loss *test set*	Waktu (Detik)
1	AlexNet	0,9985	0,0048	0,9929	0,0466	0,9938	0,0197	692
2	LiteAlexNet	0,9673	0,0977	0,9699	0,0974	0,9700	0,0866	73
3	LeNet-5	0,9918	0,0267	0,9839	0,0588	0,9860	0,0429	**27**
4	VGG16	**0,9989**	**0,0041**	0,9944	0,0358	**0,9951**	0,0223	7785
5	GoogleNet	0,9943	0,0179	0,9865	0,0447	0,9860	0,1860	1358
6	ResNet50	0,9896	0,0293	0,9824	0,0641	0,9800	0,0627	5159
7	Modifikasi VGG16	0,9968	0,0110	**0,9984**	**0,0070**	0,9936	**0,0181**	881

Berdasarkan hasil proses *training* dan *testing* model pada penelitian ini akan menggunakan model arsitektur Modifikasi VGG16 dikarenakan memiliki nilai akurasi dan *loss* yang sangat baik pada *over sampling* dan optimizer yang berbeda – beda. Modifikasi VGG16 memiliki nilai *validation set* terbaik pada *sample size* data besar, *sample size* data sedang, *sample size* data kecil, RMSprop, dan Adam. Modifikasi VGG16 juga memiliki nilai *test set* terbaik pada *sample size* data kecil (*loss)*, RMSprop (akurasi dan *loss)*, dan Adam (*loss)*. Sedangkan VGG16 memiliki nilai *training set* terbaik pada *sample size* data besar, *sample size* data sedang, *sample size* data kecil, dan Adam. VGG16 juga memiliki nilai *test set* terbaik pada *sample size* data sedang (akurasi dan *loss)*, *sample size* data kecil (akurasi*)*, dan Adam (akurasi). Pada AlexNet, nilai akurasi dan *loss training set, validation set, dan test set terbaik* pada SGD, namun nilai akurasi masih dibawah 98%.

Pesaing terdekat dari Modifikasi VGG16 yaitu VGG16. Namun Modifikasi VGG16 memiliki waktu yang lebih cepat dalam proses *testing* model jika dibandingkan dengan VGG16, jadi untuk proses deteksi status hunian parkir akan lebih cepat jika dibandingkan dengan VGG16.

Walaupun dengan akurasi yang sangat tinggi tersebut bahkan hampir mendekati sempurna, masih terdapat hasil deteksi yang tidak sesuai dengan keadaan sebenarnya seperti

pada Gambar 4.15, terutama ketika kondisi cahaya sangat minim contohnya pada saat malam hari.



Gambar 4.15. Hasil deteksi gambar area parkir ketika malam hari

Beberapa peneliti telah menggunakan data yang sama seperti penelitian ini, diantaranya adalah penelitian dari (Amato et al. 2016). Dalam penelitiannya, data yang digunakan sama, tetapi hanya menggunakan dua posisi kamera untuk mengambil gambar area parkir. Peneliti tidak menyebutkan secara spesifik posisi kamera yang digunakan. Beberapa eksperimen dalam menguji status hunian parkir yaitu dengan menggunakan single *camera* dan *multi camera.* Hasil penelitian tersebut seperti pada Tabel 4.12 dan Tabel 4.13, dimana akurasi terbaik pada eksperimen *single camera* yaitu 99,6% dengan menggunakan data *train camera A* (*even*),data *test Camera A* (*odd*), dan model arsitektur miniAlexNet.

Tabel 4.12. Eksperimen *single camera* oleh (Amato et al. 2016)

Train	Test	Model Arsitektur	Akurasi (%)
Camera A (*even*)	Camera A (*odd*)	Mini LeNet	99,3
		Mini AlexNet	99,6
Camera A (*odd*)	Camera A (*even*)	Mini LeNet	98,2
		Mini AlexNet	99,3
Camera B (*even*)	Camera B (*odd*)	Mini LeNet	86,1

		Mini AlexNet	91,1
Camera B (*odd*)	Camera B (*even*)	Mini LeNet	84,3
		Mini AlexNet	89,8

Tabel 4.13. Eksperimen multi camera oleh (Amato et al., 2016)

Train	Test	Model Arsitektur	Akurasi (%)
Camera A	Camera B	Mini LeNet	84,3
		Mini AlexNet	86,3
Camera B	Camera A	Mini LeNet	84,2
		Mini AlexNet	90,7

Selain itu pada penelitian (Tanuwijaya and Fatichah 2020a) juga menggunakan data yang sama, tetapi pada penelitian tersebut hanya menggunakan satu posisi kamera, peneliti juga tidak menyebutkan secara spesifik posisi kamera yang digunakan. Eksperimen yang dilakukan adalah dengan membandingkan model arsitektur yang berbeda, hasil terbaik dari eksperimen tersebut yaitu ada pada model arsitektur VGG16. Hasil eksperimen seperti pada Tabel 4.14.

Tabel 4.14. Hasil Eksperimen dari penelitian (Tanuwijaya and Fatichah 2020a)

Model Arsitektur	Akurasi (%)
Lite AlexNet	98,00
Mini AlexNet	98,00
AlexNet	98,00
VGG16	99,00

Hasil dari penelitian sebelumnya dan hasil dari penelitian ini tidak dapat dibandingkan, dikarenakan jumlah data dan posisi sudut pandang gambar yang berbeda, walaupun dengan menggunakan data set yang sama.

4.2 Implementasi sistem deteksi hunian parkir

Implementasi sistem deteksi hunian parkir memiliki beberapa bagian yaitu implementasi sistem untuk melihat hasil deteksi hunian parkir, implementasi database, dan implementasi sistem untuk deteksi status hunian parkir.

4.2.1 Implementasi sistem untuk melihat hasil deteksi hunian parkir

Implementasi sistem untuk melihat hasil deteksi hunian parkir yaitu aplikasi android pada penelitian ini menggunakan Bahasa Pemrograman Java dengan IDE Android Studio. Pada aplikasi android ini menggunakan konsep MVC dimana *Model* berfungsi untuk mengatur data hasil deteksi hunian parkir, *View* untuk menampilkan *layout* denah area parkir beserta status huniannya, dan *Controller* untuk mengatur jalanya data hasil deteksi untuk ditampilkan pada *View*.

Sesuai dengan rancangan antarmuka, aplikasi android diimplementasikan dengan membuat *Controller, Model,* dan *View* dengan jumlah masing-masing delapan. Contoh *source code Model* seperti pada Gambar 4.16, contoh *source code Controller* seperti pada Gambar 4.17, dan contoh tampilan aplikasi android seperti pada Gambar 4.18.

```
package com.example.parking;

public class CameraA {
    private int area1A,area2A, area3A, area4A, area5A;

    public CameraA(){
    }
    public CameraA(int area1A, int area2A, int area3A, int area4A, int
area5A){
        this.area1A = area1A;
        this.area2A = area2A;
        this.area3A = area3A;
        this.area4A = area4A;
        this.area5A = area5A;
    }
    public int getArea1A() {
        return area1A;
    }
    public void setArea1A(int area1A) {
        this.area1A = area1A;
    }
    public int getArea2A() {
        return area2A;
    }
    public void setArea2A(int area2A) {
        this.area2A = area2A;
    }
    public int getArea3A() {
        return area3A;
    }
    public void setArea3A(int area3A) {
        this.area3A = area3A;
    }
    public int getArea4A() {
        return area4A;
    }
    public void setArea4A(int area4A) {
        this.area4A = area4A;
```

```
    }
    public int getArea5A() {
        return area5A;
    }
    public void setArea5A(int area5A) {
        this.area5A = area5A;
    }
}
```

Gambar 4.16. Contoh *source code model* aplikasi android

Pada *source code model* aplikasi android, terdapat nama kelas yang menyesuaikan dengan nama pada kamera serta dalam kelas tersebut terdapat *attribute* dengan tipe nilai *integer* untuk menampung hasil deteksi yang berupa 0 atau 1. Banyaknya *attribute* menyesuaikan jumlah tempat parkir pada suatu kamera, serta dari semua *attribute* tersebut dibuat *constructor, setter*, dan *getter* untuk mempermudah initialization kelas dan pengolahan alur data.

```
package com.example.parking;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class Activity_Aparking extends AppCompatActivity {
    private DatabaseReference mDatabase;
    private Button area1A,area2A, area3A, area4A, area5A;
    private CameraA;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity__aparking);

        mDatabase = FirebaseDatabase.getInstance().getReference();

        area1A = (Button) findViewById(R.id.area1A);
        area2A = (Button) findViewById(R.id.area2A);
        area3A = (Button) findViewById(R.id.area3A);
        area4A = (Button) findViewById(R.id.area4A);
        area5A = (Button) findViewById(R.id.area5A);

      mDatabase.child("parking").child("cameraA").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                cameraA = new CameraA();
                cameraA = dataSnapshot.getValue(CameraA.class);
```

```
                if (cameraA.getArea1A() == 1) {

area1A.setBackgroundResource(R.drawable.parking_available);
                } else {

area1A.setBackgroundResource(R.drawable.parking_nonavailable);
                }
                if (cameraA.getArea2A() == 1) {

area2A.setBackgroundResource(R.drawable.parking_available);
                } else {

area2A.setBackgroundResource(R.drawable.parking_nonavailable);
                }
                if (cameraA.getArea3A() == 1) {

area3A.setBackgroundResource(R.drawable.parking_available);
                } else {

area3A.setBackgroundResource(R.drawable.parking_nonavailable);
                }
                if (cameraA.getArea4A() == 1) {

area4A.setBackgroundResource(R.drawable.parking_available);
                } else {

area4A.setBackgroundResource(R.drawable.parking_nonavailable);
                }
                if (cameraA.getArea5A() == 1) {

area5A.setBackgroundResource(R.drawable.parking_available);
                } else {

area5A.setBackgroundResource(R.drawable.parking_nonavailable);
                }
            }
            @Override
            public void onCancelled(DatabaseError databaseError) {
                System.out.println("The read failed: " +
                                   databaseError.getCode());
            }
        });

    }

    public void toCameraB(View view) {
        Intent intent= new Intent(this,MainActivity.class);
        startActivity(intent);
    }

    public void toCameraC(View view) {
        Intent intent = new Intent(this,Activity_Cparking.class);
        startActivity(intent);
    }

    public void toCameraD(View view) {
        Intent intent = new Intent(this,Activity_Dparking.class);
        startActivity(intent);
    }
```

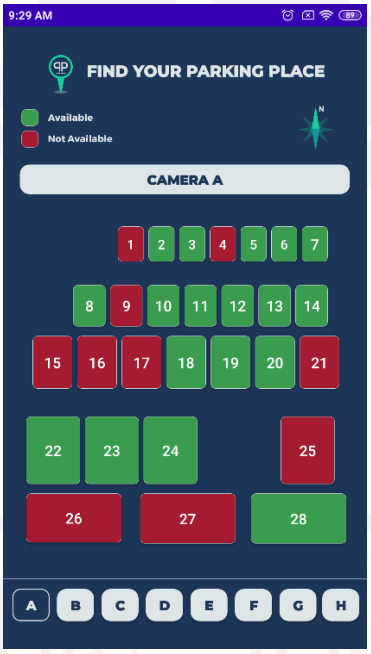
```
    public void toCameraE(View view) {
        Intent = new Intent(this,Activity_Eparking.class);
        startActivity(intent);
    }

    public void toCameraF(View view) {
        Intent intent = new Intent(this,Activity_Fparking.class);
        startActivity(intent);
    }

    public void toCameraG(View view) {
        Intent intent = new Intent(this,Activity_Gparking.class);
        startActivity(intent);
    }

    public void toCameraH(View view) {
        Intent intent = new Intent(this,Activity_Hparking.class);
        startActivity(intent);
    }
}
```

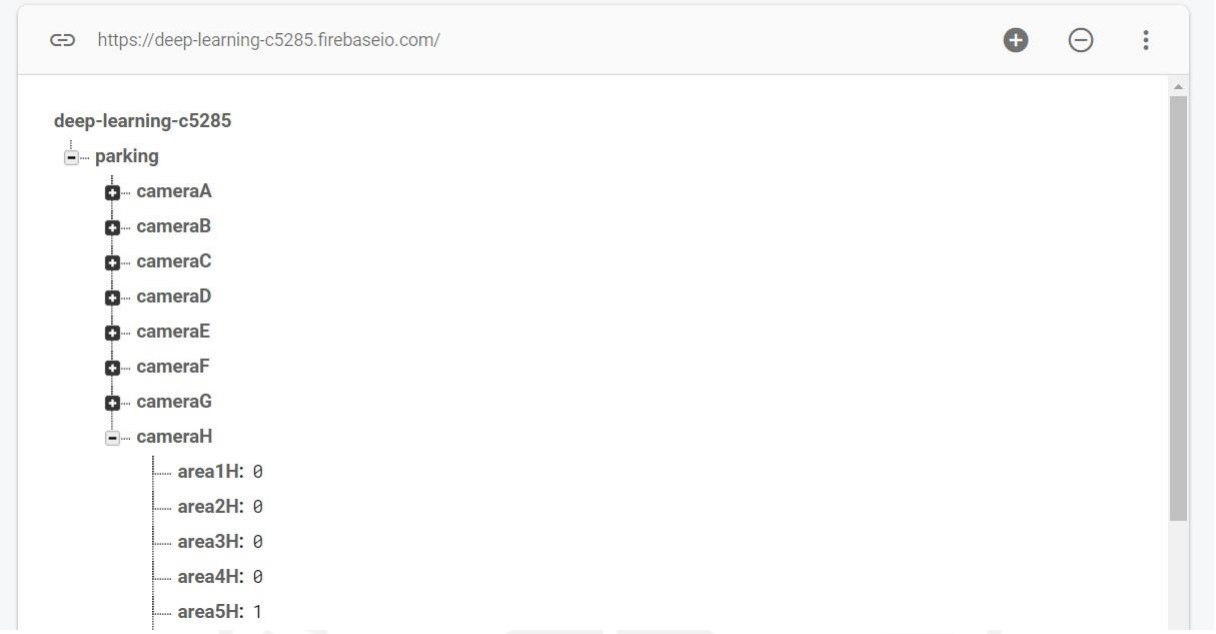
Gambar 4.17. Contoh *source code controller* aplikasi android



Gambar 4.18. Contoh tampilan aplikasi android

4.2.2 Implementasi database

Pada penelitian ini, langkah awal untuk implementasi database yaitu dengan membuat *project* pada firebase, *feature* pada firebase yang digunakan pada penelitian ini adalah *real-time database.* Bentuk struktur data pada database seperti pada Gambar 4.19, dimana struktur data tersebut hanya memuat atribut “area-beserta nomor” untuk menyimpan hasil deteksi berupa status hunian parkir yaitu kosong (1) atau penuh (0).



Gambar 4.19. Struktur database

4.2.3 Implementasi sistem untuk deteksi status hunian parkir

Tahapan implementasi sistem untuk deteksi status hunian parkir diimplementasikan menggunakan Bahasa pemrograman Pyhton dan Java.

4.2.3.1 Integrasi firebase dengan pyhton

Untuk dapat menyimpan hasil deteksi ke database, python dan firebase harus terintegrasi terlebih dahulu, dengan menggunakan *library* Pyrebase, cukup menuliskan config project firebase pada file python seperti pada Gambar 4.20, maka python dan firebase dapat terintegrasi.

```
import pyrebase

class Db:
    def save_predict(x,y,cam):
        firebase_config = {
            'apiKey': "AIzaSyBBZfSBXU9XClX_l4YocGlBoIPHq3cYVr4",
            'authDomain': "deep-learning-c5285.firebaseapp.com",
            'databaseURL': "https://deep-learning-c5285.firebaseio.com",
            'projectId': "deep-learning-c5285",
            'storageBucket': "deep-learning-c5285.appspot.com",
            'messagingSenderId': "442553609867",
            'appId': "1:442553609867:web:b54a530775114466b8ee8e",
            'measurementId': "G-E5ZXE912BT"
        }

        firebase = pyrebase.initialize_app(firebase_config)
        dbase = firebase.database()
        dbase.child('parking').child(cam).update({x:y})
```

Gambar 4.20. Integrasi firebase dengan pyhton

4.2.3.2 Import library

Beberapa *library* yang digunakan untuk mendeteksi hunian parkir seperti pada Gambar 4.21. untuk dapat menggunakan *class* db dimana *class* db digunakan untuk menyimpan hasil deteksi maka perlu *import class* tersebut terlebih dahulu.

```
import os,cv2
import pandas as pd
import numpy as np
from keras.preprocessing.image import img_to_array
from tensorflow.python.keras.models import load_model
from db import Db
```

Gambar 4.21. Import library

4.2.3.3 Membangkitkan data

Data yang digunakan seperti pada Tabel 3.1 dimana camera merupakan file csv yang berisi lokasi di setiap tempat parkirnya yang nantinya akan digunakan sebagai *bounding box* dan overlap merupakan file xlsx yang berisikan daftar slot parkir yang mengalami *overlapping* dengan slot parkir lain.

```
model_path = 'model/models_VGG16/models_modifvgg16.h5'
cameraA = pd.read_csv('./uji/cameraA.csv')
cameraB = pd.read_csv('./uji/cameraB.csv')
cameraC = pd.read_csv('./uji/cameraC.csv')
cameraD = pd.read_csv('./uji/cameraD.csv')
cameraE = pd.read_csv('./uji/cameraE.csv')
cameraF = pd.read_csv('./uji/cameraF.csv')
cameraG = pd.read_csv('./uji/cameraG.csv')
cameraH = pd.read_csv('./uji/cameraH.csv')
overlapA = pd.read_excel("./overlap/overlapA.xlsx")
overlapB = pd.read_excel("./overlap/overlapB.xlsx")
overlapC = pd.read_excel("./overlap/overlapC.xlsx")
overlapD = pd.read_excel("./overlap/overlapD.xlsx")
overlapE = pd.read_excel("./overlap/overlapE.xlsx")
overlapF = pd.read_excel("./overlap/overlapF.xlsx")
overlapG = pd.read_excel("./overlap/overlapG.xlsx")
overlapH = pd.read_excel("./overlap/overlapH.xlsx")
uji_path = './uji/ovl'
input_shape = (224,224)

#load model training
model = load_model(model_path)
```

Gambar 4.22. Membangkitkan data gambar area parkir

4.2.3.4 Prediksi status hunian parkir

Proses prediksi dilakukan secara satu per satu sesuai jumlah tempat parkir yang ada. *Variable* contours berisi lokasi – lokasi tempat parkir dimana terdapat nilai x dan y sebagai koordinat, w dan h sebagai luas dari tempat parkir. Sebuah gambar area parkir seperti pada Gambar 3.4 dipotong menjadi satu tempat parkir seperti pada Gambar 3.5 menyesuaikan nilai x, y, w, dan h. Gambar tempat parkir yang telah dipotong diubah ukurannya menjadi 224 x 224 menyesuaikan ukuran yang diperlukan sebagai input pada model. Lalu gambar tersebut diubah menjadi array yaitu dengan menggunakan *library* Numpy untuk dapat dilakukan prediksi. Hasil prediksi disimpan pada database dan membuat *bounding box* dengan warna sesuai hasil prediksi, jika hasilnya 0 maka bounding box akan berwarna merah, jika hasilnya 1 maka *bounding box* akan berwarna hijau.

```
def load_images(folder):
   l=0
   for foldername in os.listdir(folder):
       cameraName = uji_path+"/"+foldername
       l=1+l
       for filename in os.listdir(cameraName):
           if foldername == "cameraA":
              contours = cameraA
              overlap = overlapA
              cam = "A"
           elif foldername == "cameraB":
              contours = cameraB
              overlap = overlapB
              cam = "B"
           elif foldername == "cameraC":
              contours = cameraC
              overlap = overlapC
              cam = "C"
           elif foldername == "cameraD":
              contours = cameraD
              overlap = overlapD
              cam = "D"
           elif foldername == "cameraE":
              contours = cameraE
              overlap = overlapE
              cam = "E"
           elif foldername == "cameraF":
              contours = cameraF
              overlap = overlapF
              cam = "F"
           elif foldername == "cameraG":
              contours = cameraG
              overlap = overlapG
              cam = "G"
           elif foldername == "cameraH":
              contours = cameraH
              overlap = overlapH
              cam = "H"

           img = cv2.imread(os.path.join(cameraName,filename))
           if img is not None:
              n = len(contours)
              for i in contours.index:
                  #membuat bounding box

                  x = int(round(contours['X'][i] / 2.6))
                  y = int(round(contours['Y'][i] / 2.6))
                  w = int(round(contours['W'][i] / 2.6))
                  h = int(round(contours['H'][i] / 2.6))

                  #crop image dan resize
                  subpic = img[y:y + h, x:x + w]
                  subpic = cv2.resize(subpic, input_shape)

                  #prediksi
                  subpic = img_to_array(subpic)
                  subpic = np.expand_dims(subpic, axis=0)
                  answer = model.predict(subpic)
                  result = np.argmax(answer)
```

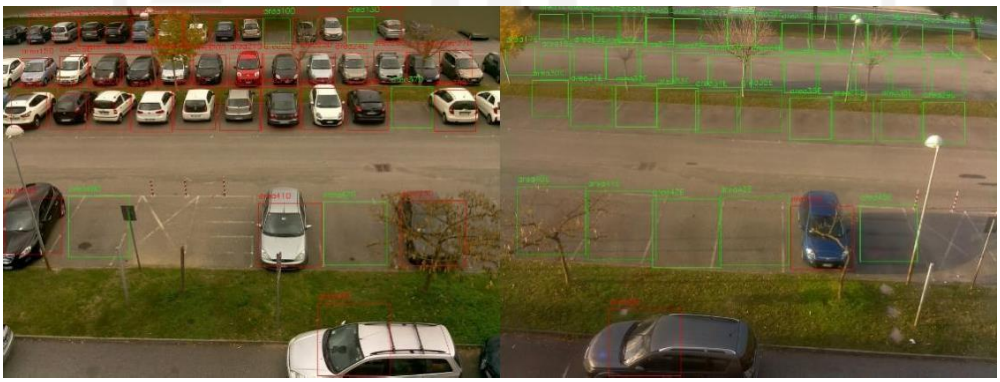
```
                    # save ke firebase
                    name = 'area' + str(n) + cam
                    print(name)
                    Db.save_predict(name, int(result), foldername)

                    #overlap
                    if pd.isna(overlap["ov1"][n-1]) != True:
                       ov1 = overlap["ov1"][n-1].astype(int)
                       cam1 = overlap["cam1"][n-1]
                       name1      =      'area'      +      str(ov1)+cam1
                       Db.save_predict(name1, int(result), "camera"+cam1)
                    if pd.isna(overlap["ov2"][n-1]) != True:
                       ov2 = overlap["ov2"][n-1].astype(int)
                       cam2 = overlap["cam2"][n-1]
                       name2    =    'area'    +    str(ov2)    +    cam2
                       Db.save_predict(name2, int(result), "camera"+cam2)

                       #memberi warna pada bounding box
                       if result == 1:
                          img = cv2.rectangle(img, (x, y), (x + w, y + h),
                                (5, 252, 5), 1)
                          img = cv2.putText(img, name, (x, y - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5,(5,252,5),1)
                       else:
                          img = cv2.rectangle(img, (x, y), (x + w, y + h),
                                (0,0, 255), 1)
                          img = cv2.putText(img, name, (x, y - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0,0,255),1)
                       n = n - 1

            cv2.imshow('img', img)
            cv2.waitKey(0)
load_images(uji_path)
```

Gambar 4.23. Deteksi status hunian parkir dari suatu area parkir



Gambar 4.24. Hasil deteksi status hunian parkir

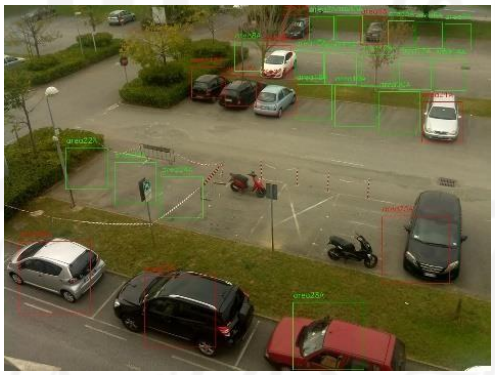
43 Pengujian

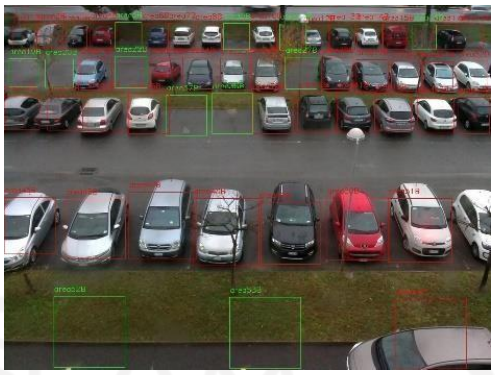
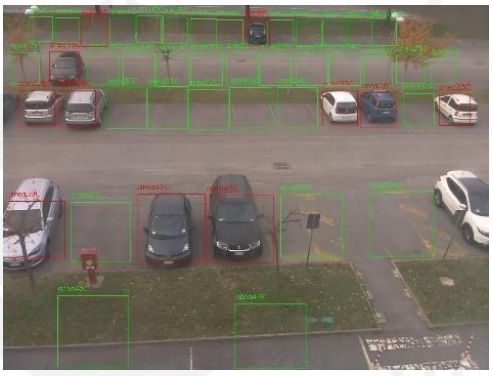
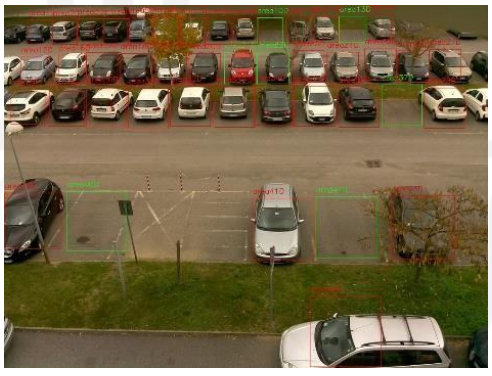
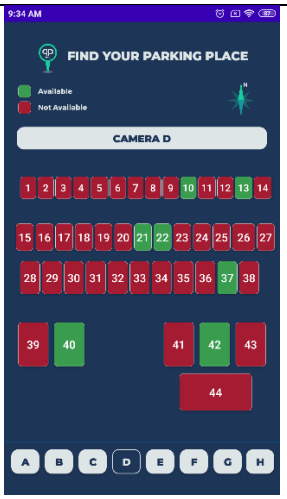
4.3.1 Pengujian simulasi sistem deteksi hunian parkir

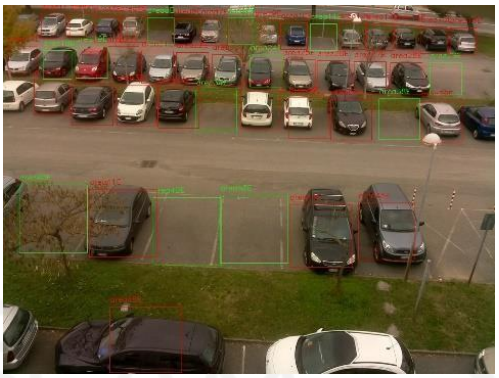
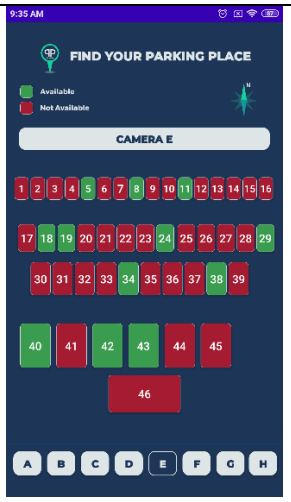
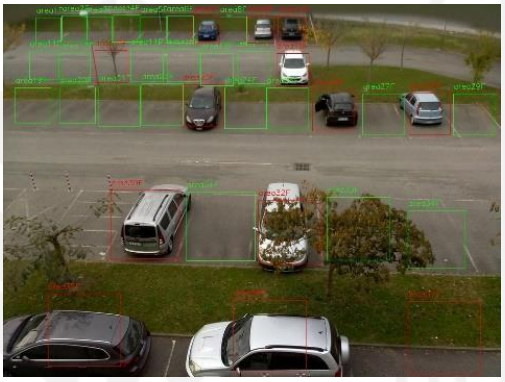
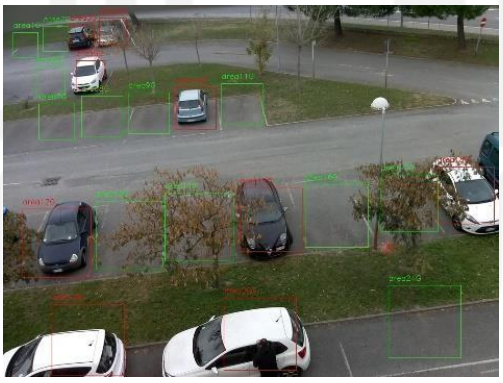
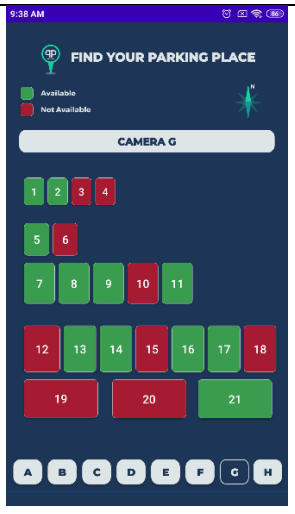
Pengujian simulasi sistem deteksi parkir dengan menggunakan metode *Black box testing* seperti pada Tabel 4.15. Pengujian ini dilakukan dengan menyiapkan 8 gambar area parkir dari kamera A sampai kamera H, lalu satu persatu dilakukan deteksi sehingga pada sistem untuk deteksi status hunian parkir akan menghasilkan *output* gambar area parkir dengan *bounding box* berwarna hijau atau merah, lalu pada sistem untuk melihat hasil deteksi parkir yaitu aplikasi android akan menghasilkan denah area parkir berwarna hijau atau merah , dari kedua *output* sub-sistem tersebut dicocokan hasilnya.

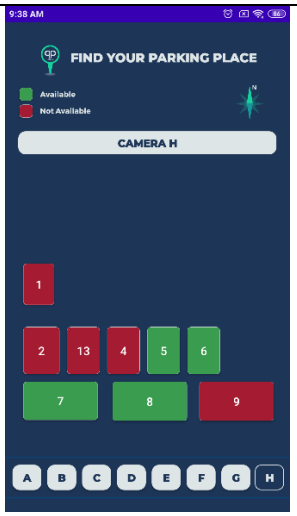
Hasil dari pengujian tersebut yaitu sangat baik, dimana setiap hasil deteksi pada gambar area parkir dapat digambarkan dengan baik oleh denah area parkir pada aplikasi android. Jika dilakukan deteksi gambar area parkir baru, maka hasil deteksi tersebut langsung dapat dilihat pada aplikasi android.

Tabel 4.15. Pengujian simulasi sistem deteksi hunian parkir

No.	Gambar area parkir	Gambar Hasil deteksi	Denah area parkir pada aplikasi android
1			

2			 9:30 AM FIND YOUR PARKING PLACE Available Not Available CAMERA B 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 A B C D E F G H
3			 9:32 AM FIND YOUR PARKING PLACE Available Not Available CAMERA C 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 A B C D E F G H
4			 9:34 AM FIND YOUR PARKING PLACE Available Not Available CAMERA D 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 A B C D E F G H

5			 9:35 AM FIND YOUR PARKING PLACE Available Not Available CAMERA E 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 A B C D E F G H
6			 9:37 AM FIND YOUR PARKING PLACE Available Not Available CAMERA F 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 37 29 30 31 32 33 34 35 36 37 A B C D E F G H
7			 9:38 AM FIND YOUR PARKING PLACE Available Not Available CAMERA G 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 A B C D E F G H

8			

4.3.2 Pengujian Usability

Pengujian usability dalam bentuk kuesioner pada penelitian ini terdiri dari tujuh pertanyaan yang disebarkan pada 24 responden dengan kritetria responden adalah pengguna kendaraan mobil. Enam pertanyaan kuesioner dibuat menggunakan skala *Likert* yaitu dari skala satu sampai lima dan satu pertanyaan dengan jawaban singkat. Skala *Likert* adalah metode perhitungan yang digunakan untuk keperluan riset atas jawaban setuju atau tidaknya seorang responden terhadap suatu pertanyaan.

Pengujian usability ini dilakukan dengan melakukan *running* pada sistem untuk deteksi status hunian parkir, kemudian responden mencoba aplikasi android untuk melihat hasilnya, dimana status hunian parkir pada aplikasi android akan berubah yang direpresentasikan oleh warna hijau atau merah.

Untuk menghitung skor maksimum setiap jawaban yaitu dengan mengalikan skor keseluruhan responden seperti pada Tabel 4.16 .

Tabel 4.16. Skor maksimum

Jawaban	Skor	Skor Maksimum (Skor * Jumlah Responden)
Sangat Setuju	5	120
Setuju	4	96
Cukup Setuju	3	72
Kurang Setuju	2	48
Tidak Setuju	1	24

Setelah itu, dapat dicari persentase masing-masing jawaban menggunakan rumus:

$$Y = \frac{total\ skor}{skor\ maksimu}\ x\ 100\%$$

Kriteria skor untuk persentase dapat dilihat pada Tabel 4.17.

Tabel 4.17. Kriteria skor

Kategori	Keterangan
0%-20%	Tidak Setuju
21%-40%	Kurang setuju
41%-60%	Cukup setuju
61%-80%	Setuju
81%-100%	Sangat setuju

Berikut ini adalah hasil persentase masing – masing jawaban yang sudah dihitung nilainya. Kuesioner ini telah diujikan kepada 24 orang responden.

1. Pertanyaan pertama

   Apakah aplikasi deteksi hunian parkir mudah untuk dipelajari dan digunakan ?

   Hasil kuesioner pertanyaan pertama dapat dilihat pada Tabel 4.18.

Tabel 4.18. Hasil kuesioner pertanyaan pertama

Pertanyaan	Jawaban	Skor	Responden	Jumlah skor	Nilai persentase (%)
1	Sangat setuju	5	15	75	$\frac{109}{120}x100\%$ = 90,83%
	Setuju	4	7	28	
	Cukup setuju	3	2	6	
	Kurang setuju	2	0	0	
	Tidak setuju	1	0	0	
Jumlah			24	109	

Berdasarkan nilai persentase dari pertanyaan pertama, dapat disimpulkan sebanyak 90,83% responden menyatakan sangat setuju bahwa aplikasi deteksi hunian parkir mudah untuk dipelajari dan digunakan.

2. Pertanyaan kedua

Apakah aplikasi deteksi parkir sudah baik untuk menggambarkan kondisi dari sebuah area parkir?

Hasil kuesioner pertanyaan kedua dapat dilihat pada Tabel 4.19.

Tabel 4.19. Hasil kuesioner pertanyaan kedua

Pertanyaan	Jawaban	Skor	Responden	Jumlah skor	Nilai persentase (%)
2	Sangat setuju	5	14	70	$\frac{111}{120}x100\%$ = 92,5%
	Setuju	4	8	32	
	Cukup setuju	3	3	9	
	Kurang setuju	2	0	0	
	Tidak setuju	1	0	0	
Jumlah			24	111	

Berdasarkan nilai persentase dari pertanyaan kedua, dapat disimpulkan sebanyak 92,5% responden menyatakan sangat setuju bahwa aplikasi deteksi parkir sudah baik untuk menggambarkan kondisi dari sebuah area parkir.

3. Pertanyaan ketiga

Apakah dengan adanya aplikasi deteksi parkir dapat memberikan bantuan informasi mengenai status hunian pada suatu area parkir?

Hasil kuesioner pertanyaan ketiga dapat dilihat pada Tabel 4.20.

Tabel 4.20. Hasil kuesioner pertanyaan ketiga

Pertanyaan	Jawaban	Skor	Responden	Jumlah skor	Nilai persentase (%)
3	Sangat setuju	5	18	90	$\frac{113}{120}x100\%$ = 94,16%
	Setuju	4	5	20	
	Cukup setuju	3	1	3	
	Kurang setuju	2	0	0	
	Tidak setuju	1	0	0	
Jumlah			24	113	

Berdasarkan nilai persentase dari pertanyaan ketiga, dapat disimpulkan sebanyak 94,16% responden menyatakan sangat setuju bahwa dengan adanya aplikasi deteksi parkir dapat memberikan bantuan informasi mengenai status hunian pada suatu area parkir.

4. Pertanyaan keempat

Menurut anda apakah aplikasi deteksi hunian parkir dapat mempersingkat waktu dalam mencari tempat parkir yang kosong?

Hasil kuesioner pertanyaan keempat dapat dilihat pada Tabel 4.21.

Tabel 4.21. Hasil kuesioner pertanyaan keempat

Pertanyaan	Jawaban	Skor	Responden	Jumlah skor	Nilai persentase (%)
4	Sangat setuju	5	18	90	$\frac{113}{120}x100\%$ = 94,16%
	Setuju	4	5	20	
	Cukup setuju	3	1	3	
	Kurang setuju	2	0	0	
	Tidak setuju	1	0	0	
Jumlah			24	113	

Berdasarkan nilai persentase dari pertanyaan keempat, dapat disimpulkan sebanyak 94,16% responden menyatakan sangat setuju bahwa aplikasi deteksi hunian parkir dapat mempersingkat waktu dalam mencari tempat parkir yang kosong.

5. Pertanyaan kelima

Warna dan icon pada aplikasi sudah sesuai?

Hasil kuesioner pertanyaan kelima dapat dilihat pada Tabel 4.22.

Tabel 4.22. Hasil kuesioner pertanyaan kelima

Pertanyaan	Jawaban	Skor	Responden	Jumlah skor	Nilai persentase (%)
5	Sangat setuju	5	12	60	$\frac{105}{120}x100\%$ = 87,5%
	Setuju	4	9	36	
	Cukup setuju	3	3	9	
	Kurang setuju	2	0	0	
	Tidak setuju	1	0	0	

Jumlah	24	105	

Berdasarkan nilai persentase dari pertanyaan kelima, dapat disimpulkan sebanyak 87,5% responden menyatakan sangat setuju bahwa warna dan icon pada aplikasi sudah sesuai.

6. Pertanyaan keenam

Jika di masa yang akan datang aplikasi deteksi parkir ini sudah bisa digunakan, apakah Anda akan sering menggunakannya?

Hasil kuesioner pertanyaan keenam dapat dilihat pada Tabel 4.23.

Tabel 4.23. Hasil kuesioner pertanyaan keenam

Pertanyaan	Jawaban	Skor	Responden	Jumlah skor	Nilai persentase (%)
5	Sangat setuju	5	17	85	$\frac{111}{120}x100\%$ = 92,5%
	Setuju	4	5	20	
	Cukup setuju	3	2	6	
	Kurang setuju	2	0	0	
	Tidak setuju	1	0	0	
Jumlah			24	111	

Berdasarkan nilai persentase dari pertanyaan keenam, dapat disimpulkan sebanyak 92,5% responden menyatakan sangat setuju bahwa jika di masa yang akan datang aplikasi deteksi parkir ini sudah bisa digunakan maka anda akan sering menggunakannya.

7. Pertanyaan ketujuh

Saran untuk pengembangan aplikasi ini ? (optional)

Pertanyaan ini bersifat *optional* dimana tidak semua responden mengisi saran untuk pengembangan aplikasi ini. Hasil kuesioner pertanyaan ketujuh sebagai berikut :

a. Ditambahkan informasi tambahan seperti waktu, suhu, arah.
b. Ditambahkan untuk area parkir roda dua
c. Untuk UI / UX bisa dikembangkan lagi
d. Penambahan fitur *booking* dan fitur *direction* (Penunjuk arah)

e. Ditambahkan fitur informasi metode pembayaran pada suatu area parkir, untuk memperkaya manfaat dari aplikasi ini. Contoh, ada beberapa area parkir yg hanya menerima pembayaran dengan uang elektronik (e-Money) dan lain - lain.

Hasil dari setiap pertanyaan satu sampai enam dilakukan perhitungan rata-rata secara keseluruhan. Kemudian akan dibandingkan seperti pada Tabel 4.24 untuk diambil kesimpulan.

Tabel 4.24. Pengolahan skala kuesioner

Pertanyaan	Nilai persentase	Keterangan
1	90,83%	Sangat Setuju
2	92,5%	Sangat Setuju
3	94,16%	Sangat Setuju
4	94,16%	Sangat Setuju
5	87,5%	Sangat Setuju
6	92,5%	Sangat Setuju
Total persentase	551,65	Sangat Setuju
Rata-rata	91,94	

Hasil dari pengujian usability dalam bentuk kuesioner diperoleh bahwa aplikasi sistem deteksi hunian parkir dalam hal kemudahan bagi pengguna untuk memahami bagaimana menggunakan antarmuka aplikasi, tingkat dukungan yang disediakan aplikasi seperti mempermudah mendapatkan informasi mengenai status hunian parkir, dan tingkat kepuasan calon pengguna akan hadirnya aplikasi ini sudah sangat baik. Sehingga secara umum aplikasi ini dapat dijadikan sebagai pengembangan sistem untuk memvisualisasikan hasil deteksi hunian parkir.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada penelitian ini dapat diperoleh kesimpulan sebagai berikut:

a. Metode *Convolutional Neural Network* (CNN) dapat diaplikasikan dengan baik untuk mendeteksi hunian parkir dari suatu gambar area parkir.
b. Dari ketujuh model arsitektur CNN, hasil akurasi dan *loss* terbaik yang dipilih oleh peneliti adalah modifikasi VGG16 dengan akurasi pada *training set* sebesar 99,75% dan loss sebesar 0,0090, sedangkan pada *validation set* akurasi yang didapat yaitu 99,89% dan loss sebesar 0,0045, serta pada *test set* hasil akurasinya adalah 99,09% dan loss sebesar 0,0365. Walaupun dengan akurasi yang sangat tinggi tersebut bahkan hampir mendekati sempurna, masih terdapat hasil deteksi yang tidak sesuai dengan keadaan sebenarnya, terutama ketika kondisi cahaya sangat minim contohnya pada saat malam hari.
c. Deteksi gambar sebuah area parkir menggunakan model yang sudah dilatih sebelumnya dapat divisualisasikan dengan sangat baik pada aplikasi android.

5.2 Saran

Pada penelitian ini masih memiliki banyak kekurangan yang perlu diperbaiki sehingga membutuhkan saran untuk pengembangan pada penelitian selanjutnya. Berikut saran yang dapat dipertimbangkan bagi penelitian selanjutnya:

a. Pada penelitian ini hanya menggunakan data berupa gambar area parkir, diharapkan pada penelitian selanjutnya dapat menggunakan data berupa video khususnya area parkir di Indonesia, supaya hasil deteksi dapat dipantau secara langsung.
b. Dalam penelitian ini masih sebatas menggunakan model arsitektur CNN yang sudah ada dan waktu yang dibutuhkan untuk proses training cukup lama. Untuk itu penulis berharap penelitian ini dapat dikembangkan dengan melakukan improvisasi pada model arsitektur atau bahkan menemukan metode baru yang lebih efektif dan efisien dengan GPU dan CPU yang lebih baik.
c. Penulis berharap penelitian ini dapat dijadikan referensi untuk mengimplementasikan sistem deteksi hunian parkir

DAFTAR PUSTAKA

Acharya, Debaditya, Weilin Yan, and Kourosh Khoshelham. 2018. "Real-Time Image-Based Parking Occupancy Detection Using Deep Learning." *CEUR Workshop Proceedings* 2087:33–40.

Ahmad, Abu. 2017. "Mengenal Artificial Intelligence, Machine Learning, Neural Network, Dan Deep Learning." *Jurnal Teknologi Indonesia* (June).

Akbar, Muhammad, and Suwatri Jura. 2019. "Sistem Informasi Realtime Web Untuk Slot Parkir Berbasis Embedded System." *Jurnal IKRA-ITH Informatika* 3(2):33–38.

Alamsyah, Derry. 2017. "Pengenalan Mobil Pada Citra Digital Menggunakan." *ISSN: 1978-1520* 162–68.

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 2007. "Handbook of Approximation Algorithms and Metaheuristics." *Handbook of Approximation Algorithms and Metaheuristics* 1–1432. doi: 10.1201/9781420010749.

Amato, Giuseppe, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, and Claudio Vairo. 2016. "Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning." *Proceedings - IEEE Symposium on Computers and Communications* 2016-Augus(Dl):1212–17. doi: 10.1109/ISCC.2016.7543901.

Anon. n.d. "Pengenalan Deep Learning Part 7 : Convolutional Neural Network (CNN) | by Samuel Sena | Medium." Retrieved November 8, 2020 (https://medium.com/@samuelsena/pengenalan-deep-learning-part-7-convolutional-neural-network-cnn-b003b477dc94).

Camacho, Cezanne. 2018. "Convolutional Neural Networks – Cezanne Camacho – Machine and Deep Learning Educator."

Dewisita, Nurprastia Amanda, Nuryanto Nuryanto, and Auliya Burhanuddin. 2019. "Prototype Sistem Pengelolaan Parkir Dengan Sensor Ldr (Light Dependent Resistor) Untuk Optimalisasi Layanan Tempat Parkir Mobil." *Jurnal Komtika* 2(2):124–28. doi: 10.31603/komtika.v2i2.2599.

Eka Putra, Wayan Suartika. 2016. "Klasifikasi Citra Menggunakan Convolutional Neural Network (CNN) Pada Caltech 101." *Jurnal Teknik ITS* 5(1). doi: 10.12962/j23373539.v5i1.15696.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning

for Image Recognition." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-Decem:770–78. doi: 10.1109/CVPR.2016.90.

Ilahiyah, Sarirotul, and Agung Nilogiri. 2018. "Implementasi Deep Learning Pada Identifikasi Jenis Tumbuhan Berdasarkan Citra Daun Menggunakan Convolutional Neural Network." *Jurnal Sistem & Teknologi Informasi Indonesia* 49–56.

K, Shihabudin Achmad Muhajir A., and Safrina Amini. 2016. "Sistem Monitoring Tempat Parkir Dengan Sensor Ultrasonik Berbasis Arduino Uno Pada Cibinong City Mall." *Seniati* 350–55.

Lecun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. "Deep Learning." *Nature* 521(7553):436–44. doi: 10.1038/nature14539.

Li, Hui, and Chunhua Shen. 2016. "Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs."

Limbong, Rahmat. 2020. "Implementasi Algoritma Unary Coding Pada Kompresi Citra Ultrasonografi." *Jurnal Pelita Informatika* 8:367–70.

Lina, Qolbiyatul. 2019. "Apa Itu Convolutional Neural Network? | by QOLBIYATUL LINA | Medium."

Liu, Derrick, and Yushi Wang. 2017. "Monza: Image Classification of Vehicle Make and Model Using Convolutional Neural Networks and Transfer Learning."

Maulana, Rizqy, Hurriyatul Fitriyah, and Esa Prakasa. 2018. "Implementasi Sistem Deteksi Slot Parkir Mobil Menggunakan Metode Morfologi Dan Background Subtraction." 2(5):1954–59.

Mele, Barbara, and Guido Altarelli. 2014. "Lepton Spectra as a Measure of b Quark Polarization at LEP." *Physics Letters B* 299(3–4):345–50. doi: 10.1016/0370-2693(93)90272-J.

Nugraha, Rahadian, Agung Nugroho Jati, and Umar Ali Ahmad. 2016. "Implementasi Histogram of Oriented Gradient (HOG) Pada Embedded System Untuk Identifikasi Slot Parkir Sebagai Pendukung Smart Parking System." *E-Proceeding of Engineering* 3(Universitas Telkom):771–77.

Ola, Yusuf Tacob Ona, Suyoto, and Sigit Purnomo. 2016. "Pengujian Usability Antarmuka Aplikasi Mangente." *Seminar Nasional Teknologi Informasi Dan Komunikasi* 2016(Sentika):334–42.

Paidi, V., Fleyeh, H., Nyberg, R. G. 2018. "Deep Learning-Based Vehicle Occupancy

Detection in an Open Parking Lot Using Thermal Camera."

Ramadani. 2017. "Firebase Realtime Database Dengan Android | by Ramadani | Javan Cipta Solusi."

Samsiana, Seta, Rahmadya Trias Handayanto, Anita Setyowati, Srie Gunarti, Irwan Raharja, Fata Nidaul Khasanah, Jawa Barat, Fakultas Teknologi Informasi, Universitas Bina, Sarana Informatika, Karawang Barat, Jawa Barat, Fakultas Teknik, Universitas Bhayangkara, Jakarta Raya, Marga Mulya, Bekasi Utara, Fakultas Teknik, Universitas Muhamadiyah Magelang, Jawa Tengah, Fakultas Teknologi, Komunikasi Dan, Universitas Nasional, Kota Jakarta Selatan, and Daerah Khusus Ibukota. 2020. "Optimasi Penggunaan Android Sebagai Peluang Usaha Di Masa Pandemik COVID ' 19." 3(2):137–48.

Setiawan, Jemima Ciani, Resmana Lim, M. Eng, and Justinus Andjarwirawan. 2017. "Implementasi Internet of Things Untuk Parkir Mobil Dengan Pembayaran Menggunakan QR Code."

Siahaan, Mangapul, Christopher Harsana Jasa, Kevin Anderson, and Melissa Valentino. 2020. "Penerapan Artificial Intelligence (AI) Terhadap Seorang Penyandang Disabilitas Tunanetra." *Journal of Information System and Technology* 01(02):1–8.

Sofia, Nadhifa. 2018. "CONVOLUTIONAL NEURAL NETWORK. Convolutional Neural Network Adalah… | by Nadhifa Sofia | Medium."

Statistik, Badan Pusat. 2018. "Badan Pusat Statistik."

Statistik, SPSS. 2018. "Data Primer Dan Sekunder | SPSS Statistik."

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. "Going Deeper with Convolutions." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 07-12-June:1–9. doi: 10.1109/CVPR.2015.7298594.

Tanuwijaya, Evan, and Chastine Fatichah. 2020a. "Modification of Alexnet Architecture for Detection of Car Parking Availability in Video Cctv." *Jurnal Ilmu Komputer Dan Informasi* 13(2):47–55. doi: 10.21609/jiki.v13i2.808.

Tanuwijaya, Evan, and Chastine Fatichah. 2020b. "Penandaan Otomatis Tempat Parkir Menggunakan YOLO Untuk Mendeteksi Ketersediaan Tempat Parkir Mobil Pada Video CCTV." *Briliant: Jurnal Riset Dan Konseptual* 5(1):189. doi: 10.28926/briliant.v5i1.434.

Thakur, Rohit. 2019. "Step by Step VGG16 Implementation in Keras for Beginners | by Rohit Thakur | Towards Data Science."

Thomas, Tom, and Tarun Bhatt. 2018. “Smart Car Parking System Using Convolutional Neural Network.” *Proceedings of the International Conference on Inventive Research in Computing Applications, ICIRCA 2018* (Icirca):172–74. doi: 10.1109/ICIRCA.2018.8597227.

Yakura, Hiromu, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. 2018. “Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism.” *CODASPY 2018 - Proceedings of the 8th ACM Conference on Data and Application Security and Privacy* 2018-Janua(September):127–34. doi: 10.1145/3176258.3176335.

Zahid, Muhammad Zuhair. 2018. “Aplikasi Berbasis Android Untuk Pembelajaran : Potensi Dan Metode Pengembangan.” *PRISMA. Prosiding Seminar Nasional Matematika* 1:910–18.

Zhang, Lin, Junhao Huang, Xiyuan Li, and Lu Xiong. 2018. “Vision-Based Parking-Slot Detection: A DCNN-Based Approach and a Large-Scale Benchmark Dataset.” *IEEE Transactions on Image Processing* 27(11):5350–64. doi: 10.1109/TIP.2018.2857407.

Zhang, Xiangyu, Jianhua Zou, Kaiming He, and Jian Sun. 2016. “Accelerating Very Deep Convolutional Networks for Classification and Detection.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38(10):1943–55. doi: 10.1109/TPAMI.2015.2502579.

Zulkarnain, Dikki, and Engelin Shintadewi Julian. 2017. “Perancangan Sistem Parkir Dengan Rekomendasi Lokasi Parkir.” *JETri Jurnal Ilmiah Teknik Elektro* 14(2):17–28.

LAMPIRAN